

This Page Is Inserted by IFW Operations
and is not a part of the Official Record

BEST AVAILABLE IMAGES

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images may include (but are not limited to):

- BLACK BORDERS
- TEXT CUT OFF AT TOP, BOTTOM OR SIDES
- FADED TEXT
- ILLEGIBLE TEXT
- SKEWED/SLANTED IMAGES
- COLORED PHOTOS
- BLACK OR VERY BLACK AND WHITE DARK PHOTOS
- GRAY SCALE DOCUMENTS

IMAGES ARE BEST AVAILABLE COPY.

**As rescanning documents *will not* correct images,
please do not report the images to the
Image Problem Mailbox.**

Sun Aug 9 10:33:18 1998

Listing for Adam Sartel

```

* Revision 1.24 1998/04/26 13:45:25 ariels
* Fixed lg2() implementations on platforms where it isn't built-in to
* give identical results on all platforms. Can use double-precision
* here since lg2() is only called for initialisation.
*
* Revision 1.23 1998/04/21 06:34:09 ariels
* Dirty letters were being IGNORED in cprof_node_char() due to a
* premature semicolon terminating the scor: expression!!
*
* Revision 1.22 1998/04/15 14:55:44 ariels
* Support "very clean" sequences in cprof. These differ from dirty
* locations in that the entire sequence in the multiple alignment is
* assumed to be very clean (and no positions in it may be dirty).
*
* Revision 1.21 1998/04/15 09:25:03 ariels
* New version of cprof_node_char() deals correctly with dirty letters.
*
* Revision 1.20 1998/04/14 09:43:27 ariels
* New format for multiple alignment to allow "dirty" positions.
*
* Revision 1.19 1998/04/08 15:28:58 ariels
* Unify 1.18.1 branch (actually identical to its tip); use with cprof.c
* rev. >= 1.15 and cprof.h rev. >= 1.11.
*
* Revision 1.18.1.5 1998/03/04 07:04:38 ariels
* Fix bug in alignment when both profiles and all node sums were <32, but
* some node+link sums (for gaps) were >=32 -- the wrong scoring routine
* was used!
*
* Revision 1.18.1.4 1998/03/02 15:38:31 ariels
* Eliminate scanning in improve_cprof(), which should improve performance.
*
* Revision 1.18.1.2 1998/02/26 20:21:35 ariels
* Increase MAX_MEM to use quadratic space for far larger alignments
*
* Revision 1.18.1.1 1998/02/23 14:26:42 ariels
* Klugey 'N'-counting code.
* Doesn't affect the alignments themselves, but lies successfully about
* the results.
*
* Revision 1.18 1998/02/17 16:18:27 ariels
* Make cprof_print print counted profiles in a format which
* better-resembles other profiles. Also use cprof_num_print, which is
* like cprof_print but writes prefixed lines to a file pointer with
* positions not counted from 0.
*
* Revision 1.17 1998/02/08 10:44:52 ariels
* Eliminate debugging messages from stdout.
*
* Revision 1.16 1998/02/04 16:36:27 ariels
* Change identity %age calculation slightly.
*
* Fix bug which caused incorrect score calculation in squish_lgaps().
* YOU MUST USE REVISION 1.6 (AT LEAST) OF AC_UPDATE.K TO COMPILE!
*
* Revision 1.15 1998/02/01 12:17:00 ariels
* Add local improvements for cprof_update()
*
* Revision 1.14 1998/01/20 09:35:18 ariels
* Fix cprof_print to use correct types in printf() statements.

```

align_cprof.c

Sun Aug 9 10:33:18 1998

Listing for Adam Sartel

```

/*
* align_cprof.c -- 6-state, 18-transition model, extended to counted
* profiles
*
* This is the "normal" dynamic-programming for Smith-Waterman with long
* gaps, with some changes. Most importantly, we calculate scores using
* a maximal-additional-likelihood model, by comparing the likelihood of
* the proposed result to the product of the likelihoods of the two
* components.
*
* There is no "long-gap-extend" penalty, but it should be easy to add
* one. Should a "free" transition from long X-gap to long Y-gap be
* allowed?
*
* Contains both quadratic- and linear-space routines. Should we also
* have multi-hit here?
*
* Returns a variant (of course) of align_data: a list of all
* "normally"-aligned domains (i.e. Smith-Waterman type, as opposed to
* "x-gap/y-gap/alt zones"); for each domain we list the starting and
* ending positions in each profile, and the list of (Smith-Waterman
* type) states along that portion of the alignment.
*
* Raveh should be able to recognise lots of this code as ripped off
* from dp.c. Should there be greater unity in the code?
*
* "Klugey"-code-includes" are used to keep a single copy of the
* matrix-update code, even though it's used 3 times. See ac_update.k.
*
* ariels@compugen, 19/10/97
*
* $Log: align_cprof.c,v $
* Revision 1.29 1998/06/25 14:12:48 ariels
* Fix "BUG" in squish_lgaps() (!!) -- Last alignment position was
* updated correctly only when the last alignment operation was 'match'.
*
* Revision 1.28 1998/06/24 14:01:20 ariels
* Determine memory use of alignment (quadratic-space limit and
* log-likelihood paging table) according to machine specifics.
*
* Revision 1.27 1998/06/18 14:42:47 ariels
* Strip "#ifdef CPROF_ALIGN" and "#ifdef IMPROVE_CPROF" lines.
*
* Revision 1.26 1998/06/15 06:52:00 ariels
* Merge improvements to main trunk
*
* Revision 1.25.1.3 1998/06/15 06:20:28 ariels
* Much improved (faster) local improvements; other speed improvements.
*
* Revision 1.25.1.2 1998/05/13 10:33:09 ariels
* Add an "inline" hint; fix bug in improve_cprof() when p->len == WINDOW_SIZE
*
* Revision 1.25.1.1 1998/05/06 07:48:44 ariels
* Page the table (and treat cprofs up to width 127 faster)
*
* Revision 1.25 1998/05/04 14:30:49 ariels
* Ignore cprofs shorter than WINDOW_SIZE in improve_cprof().

```

align_cprof.c

```

* Use better constants in create_align_prm.
*
* Revision 1.13 1998/01/11 14:39:26 ariels
* Fix NULL pointer bug in cprof_node_char.
*
* Revision 1.12 1998/01/11 14:09:34 ariels
* idiotic bugfix
*
* Revision 1.11 1998/01/11 13:57:42 ariels
* Add cprof_calc_id_sim() to calculate the %age of identity and
* similarity in a given alignment. See code for exact definition of
* identity and similarity in cprof_alignments.
*
* Revision 1.10 1998/01/11 07:37:01 ariels
* Ensure sizeof(score_t) == 4. This should be done at compile-time
* somehow, but that's impossible. Instead, check it when
* create_align_prm() is called.
*
* Revision 1.9 1998/01/08 13:24:35 ariels
* Make parameter block (prm) static in align_cprof.c, rather than an
* externally supplied set of values.
*
* Move cprof_print() and cprof_compute_assembly() to here, since they
* now depend on the static parameter block. These new versions of the
* routines should be better, since they give the maximal-likelihood
* letter (which is used by the algorithm itself), rather than relying on
* some arbitrary constants to decide if something is a letter or a gap.
*
* Revision 1.8 1998/01/04 09:51:48 ariels
* Fixed fixed-point underflow bug (-MAXINT-2 is a 'large' *positive*
* integer, so all global alignments tend to fail; since all linear-space
* alignments use global alignments, they all failed).
*
* Revision 1.7 1997/12/31 13:18:46 ariels
* Add long gap-extend type transition from xgaps to ygaps. Do we really
* want this? Maybe we want to pay long gap-open for this?
*
* Revision 1.6 1997/12/31 10:34:46 ariels
* Removed 'cost' field from profiles; instead of it are a pair of static
* variables in align_cprof.c which hold the location scores for the pair
* of profiles being aligned.
*
* Revision 1.5 1997/12/30 12:40:50 ariels
* Store all scores internally in fixed-point instead of floating-point
* format. Loses some accuracy for low-magnitude scores, but ensures
* that backtracking will always be exact (and thus successful).
*
* Must have 32-bit ints. How can we ensure this at compile-time?
*
* Revision 1.4 1997/12/17 10:16:48 ariels
* Wrote linear-space version.
*
* Fixed bugs relating to asymmetry of entering/leaving long gaps.
*
*
static char rcsid[] = "$Id: align_cprof.c,v 1.29 1998/06/25 14:12:48 ariels Exp
$";

```

align_cprof.c

```

/* NULL/malloc()/free() /... */

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#include <assert.h>
#include <math.h>
#include "error.h"

#ifdef PI
#define PI 3.1415926535897932385
#endif
#include "cprof.h"
#include "cpof_align.h"
#include "align_prm.h"

/* Not all platforms define a log2() function, so we provide one here... */
#ifdef _alpha
#define lg2(x) log2f(x)
#else
static float lg2f(float x)
{
    return (float) log((double) x);
}
#endif
static float my_lg2(float x)
{
    return (float) (log(x)*M_LOG2E);
}
#define lg2(x) my_lg2(x)
#endif

enum mat_state {
    e_all=0,
    e_lgap1,
    e_lgap2,
    /* e_alt,
    n_mat_states
    };

/* Score by... */
/* LikeLihood */
/* x-gap -- Long GAP in #1 */
/* y-gap -- Long GAP in #2 */
/* Alternate sequence */

/* 3 types of matrix nodes:
* mat_node is used for quadratic alignments, and contains only a
* score.
* lmat_node is used for finding the bounding box of a linear

```

align_cprof.c

```

* alignment, and contains a score and a starting position.
*
* * llmat_node is used for finding where the optimal path crosses the
* * halfway marks, and contains a score and 2 starting position.
*
* * The same code is #included from a file 3 times to perform the score
* * updates. Different macros need to be defined each time to update
* * the correct parts of the structure. However, the score field
* * (named 'sc') must always be present, since it is used by the
* * included code!
*
*
*
* * This should probably be a datatype with sizeof(int32) == 4. Also,
* * changing it doesn't really work (you need to do fixed-point arithmetic
* * stuff separately.
*
*
* #ifdef USE_FLOAT_SCORE
/* Use more accurate scores (floats) with less accurate comparisons */
typedef float score_t;
#define FLT2SCORE(x) (x)
#define SCORE2FLT(x) (x)
#define EQ(a,b) ((fabs((a)-(b))<5e-2)
#define NaN(x) (isnanf(x))
#define NEG_INFNTY -FLT_MAX
#else
/* (DEFAULT) Use less accurate scores with exact comparisons */
#include <limits.h>
#define INT_MAX == 2147483647L
typedef int score_t;
#define LONG_MAX == 2147483647L
typedef long score_t;
#else
#error "Failed to defined a 4-byte integer type for score_t"
#endif
/* SCALE units of score_t = 1 unit of score */
#define SCALE 16384.0
#define FLT2SCORE(x) ((score_t)((x)*SCALE+0.5))
#define SCORE2FLT(x) ((float)(x)/SCALE)
#define EQ(a,b) ((a)==(b))
#define NaN(x) ((x)==0x80000000)
#define NEG_INFNTY -(long)(1<<30)
#endif

```

```

typedef struct (
    score_t sc;
    mat_node[n_mat_states];
) mat_node[n_mat_states];

typedef struct (
    score_t sc;
    int i0, j0;
    llmat_node[n_mat_states];
) llmat_node[n_mat_states];

typedef struct (
    score_t sc;
    int i0, j0;
    llmat_node[n_mat_states];
) llmat_node[n_mat_states];

```

align_cprof.c

```

/* Maximal memory (in bytes) to take when running in quadratic-space mode */
/* (should be set in main program, this is just in case...) */
#ifdef DEBUG_LINEAR
long cprof_align_max_mem = 900*sizeof(mat_node);
#else
long cprof_align_max_mem = (1<<20)*16*sizeof(mat_node);
#endif

static score_t *cost1, *cost2; /* current costs of profile positions */

static align_prm prm;

struct rect_s {
    int i0, j0, i1, j1;
};

struct match_state_s {
    char *p; /* current alignment char */
    int i, j; /* location in matrix */
    enum mat_state s; /* current state */
};

struct alignment_rec_s {
    int i0, j0, i1, j1; /* Record of an optimal alignment */
    enum mat_state st; /* Alignment lives in i0 <= i <= i1,
    j0 <= j <= j1 */
};

/* The Genetic Code: Arbitrary? */
/* (gap must 'really' be 0!) */
enum e_DNA {e_gap=0, e_a, e_c, e_g, e_t};

static float H(float x)
{
    if (fabs(x) < 1e-10 ||
        fabs(1-x) < 1e-10)
        return 0.0;
    else
        return -x*lg2(x)-(1-x)*lg2(1-x);
}

static float lg_tbl[n_mat_states];

static float lg_comb(float n, float k)
{
    if (n > 0.0) {
        /* approximate with entropy */
        float eps = k/n;
        if (fabs(eps) < 1e-8 || fabs(1.0-eps) < 1e-8)
            return 0;
        else
            return n * H(eps) - 0.5*lg2(2*PI*n*eps*(1-eps));
    }
}

```

align_cprof.c

```

    else {
        if (fabs(k)<1e-8)
            return 0;
        else
            err("Bad log combinations lg_comp(%d, %d)", n, k);
    }
}

score_t log_likelihood(cprof_node A)
{
    int i;
    float best;
    int num;
    float sum;
    float hit, miss, gaps;
    float score1, score2;

    /* value of highest element */
    /* # of times best appeared in letters */
    /* # of hits, # of misses, and if
       we're looking at a letter # of gaps */

    /* Decide what should *really* have been here */
    best = -1;
    sum = 0;
    num = 0;
    for (i=1; i<N_PROFILE_LETTERS; i++) {
        sum += A[i];
        if (A[i] > best) {
            best = A[i];
            num = 1;
        }
        else if (EQ(A[i], best))
            num++;
    }
    sum += A[e_gap];

    /* Compute log-likelihood, taking highest of 2 possibilities: */

    /* Case A: we're looking at a gap, with some (mistaken) insertions */
    hit = A[e_gap];
    miss = sum - hit;

    score1 = /* lg_comb(sum, hit) + */
             /* prm.lg_no_ins + */
             /* prm.lg_ins */
             normal /*;

    /* Case B: we're looking at a letter, with some (mistaken) changes,
       and also some (mistaken) deletions (these appear as gaps). */
    hit = best;
    gaps = A[e_gap];
    miss = sum-hit-gaps;
    score2 = -2.0 + /* lg2(1/4) */ lg_tbl[num] +
             /* lg_comb(sum, hit) + */
             /* prm.lg_no_change + */
             /* prm.lg_change + */
             /* prm.lg_del */
             normal /*;

```

align_cprof.c

```

    score1 = (score1 > score2) ? score1 : score2;
    return FLT2SCORE(score1);
}

/* ----- page table for log_likelihood when all values are <128 ----- */

/*
 * Page/offset access macros
 *
 * Split counts <128 into top 4 bits (combined for page number) and
 * bottom 3 bits (combined for offset in page).
 *
 * This works out as 15 bits for offset inside a page (32K cells) and 20
 * bits to access a page (1M pages).
 */

/* Top bits (for page number) */
#define C_PAG(x) (((x) & 0x78) >> 3)
/* Bottom bits (for offset) */
#define C_OFF(x) ((x) & 0x07)

/* Combine top bits into a page number */
#define PAGE(N) ((C_PAG(N[0])) | (C_PAG(N[1]) << 4) | \
                  (C_PAG(N[2]) << 8) | (C_PAG(N[3]) << 12) | \
                  (C_PAG(N[4]) << 16))
/* Combine bottom bits into an offset into a page */
#define OFF(N) ((C_OFF(N[0])) | (C_OFF(N[1]) << 3) | \
                (C_OFF(N[2]) << 6) | (C_OFF(N[3]) << 9) | \
                (C_OFF(N[4]) << 12))

#define PG_SZ (1<<15)
#define N_PG_PTRS (1<<20)

/* Amount of memory (in bytes) to use for log-likelihood speedup tables.
 */
long cprof_align_max_tbl_sz = 1024 * PG_SZ;

/* TUNABLE PARAMETER -- # of 128Kbyte pages to use. Set according to
 * cprof_align_max_tbl_sz, which gives # of bytes to use for all parts of
 * the table.
 */
static int num_pages;

/* Illegal page number -- used to flag free pages at beginning */
#define NO_PAGE -1

/* qman should contain a "quiet" (non-signaling) IEEE754 NaN */
#if USE_FLOAT_SCORE
/* Assumes little-endian (?) */
static unsigned int qman_word = 0xffc00000;
static score_t qman = *(float*)&qman_word;
#else
static score_t qman = 0x80000000;
#endif

struct score_pg_s {

```

align_cprof.c

```

int page;
char mark : 1;
score_t tbl[PG_SZ];
};

/* SMP: This should be shared if using SMP */
struct score_pager_s {
    struct score_pg_s *pages; /* num_pages (pre-allocated) pages */
    struct score_pg_s **ptbl; /* Pointers into pages or NULL */
    int last; /* Last page swapped */
};

/* Second chance algorithm */
static int get_victim(struct score_pager_s *pg)
{
    while (pg->pages[pg->last] = (pg->last+1) % num_pages).mark)
        pg->pages[pg->last].mark = 0;
    return pg->last;
}

static void swap_page(struct score_pager_s *pg, int k, int p)
{
    int n = pg->pages[k].page;
    int i;

    printf(">>>Swap %d into victim %d\n", p, k);

    if (n != NO_PAGE)
        pg->ptbl[n] = NULL;

    pg->ptbl[p] = &(pg->pages[k]);

    for(i=0; i<PG_SZ; i++)
        pg->pages[k].tbl[i] = qnan;
    pg->pages[k].page = p;
}

#pragma inline(ll_get)
score_t ll_get(cprof_node N)
{
    int pg = PAGE(N);
    int off = OFF(N);
    struct score_pager_s *p = prm.score_pgr;
    score_t x;

    if (p->ptbl[pg] == NULL) { /* Find a page to use */
        int so = get_victim(p);
        swap_page(p, so, pg);
        p->ptbl[pg]->mark = 1;
    }

    x = p->ptbl[pg]->tbl[off];
    if (NaN(x))
        return p->ptbl[pg]->tbl[off] = log_likelihood(N);
    else
        return x;
}

```

```

struct score_pager_s *ll_init(void)
{
    struct score_pager_s *ret;
    int i;

    num_pages = (cprof_align_max_tbl_sz -
        N_PG_PTRS*sizeof(struct score_pg_s)) /
        sizeof(struct score_pg_s);
    if ((ret = malloc(sizeof(struct score_pager_s))) == NULL ||
        (ret->pages = malloc(sizeof(struct score_pg_s) * num_pages)) == NULL ||
        (ret->ptbl = malloc(sizeof(struct score_pg_s) * N_PG_PTRS)) == NULL)
        err("Memory failure in ll_init");

    for(i=0; i < num_pages; i++) {
        ret->pages[i].page = NO_PAGE;
        ret->pages[i].mark = 0;
    }
    for(i=0; i < N_PG_PTRS; i++)
        ret->ptbl[i] = NULL;

    ret->last = -1;

    return ret;
}

void create_align_prm(float change, float del, float ins,
    float clean_change, float clean_del, float clean_ins,
    float dirty_change, float dirty_del, float dirty_ins,
    float lgo)
{
    int i;

    if (sizeof(score_t) != 4) {
        #ifdef USE_FLOAT_SCORE
            err("Bad compilation -- must have IEEE *floats* (4 bytes) for score_t\n");
        #else
            err("Bad compilation -- must have 4-byte ints for score_t\n");
        #endif
        #ifdef "Bad compilation -- must have 4-byte ints for score_t\n"
            err("Bad compilation -- must have 4-byte ints for score_t\n");
        #endif
        /*UNREACHED*/
    }
    for(i=1; i<N_mat_states; i++)
        lg_tbl[i] = lg2((float)i);

    prm.lg_change = lg2(change/3.0);
    prm.lg_del = lg2(del);
    prm.lg_no_change_del = lg2(1.0-change /*/3.0*/-del);
    prm.lg_ins = lg2(ins/4.0);
    prm.lg_no_ins = lg2(1.0-ins /*/4.0*/);

    prm.lg_clean_change = lg2(clean_change/3.0);
    prm.lg_clean_del = lg2(clean_del);
    prm.lg_clean_no_change_del = lg2(1.0-clean_change /*/3.0*/-clean_del);
    prm.lg_clean_ins = lg2(clean_ins/4.0);
    prm.lg_clean_no_ins = lg2(1.0-clean_ins /*/4.0*/);
}

```

```

/*
 * Elide all long gaps from a single-node alignment list, yielding a
 * list of all the non-long-gap areas. Free the original and return
 * the result.
 */
static cprof_align squish_lgaps(cprof_align a, cprof pl, cprof p2)
{
    int i, j, k, l, m;
    int len=strlen(a->str);
    short have_block;
    short had_block;
    cprof_node N;
    cprof_align res, last;

    last = res = NULL;
    have_block = 0;
    i = a->start1;
    j = a->start2;
    if (a->str[0] == ALIGN_LGAP1)
        i++;
    if (a->str[0] == ALIGN_LGAP2)
        j++;
    for(k = 0; k<len; k++) {
        switch(a->str[k]) {
            case ALIGN_MATCH:
            case ALIGN_GAP1:
            case ALIGN_GAP2:
                if (! have_block) {
                    if (last == NULL) {
                        if ((res = malloc(sizeof(*res))) == NULL)
                            err("memory failure for head in squish_lgaps");
                        last = res;
                    }
                    else {
                        if ((last->next = malloc(sizeof(*res))) == NULL)
                            err("memory failure in squish_lgaps");
                        last = last->next;
                    }
                    if ((last->str = malloc(len-k+1)) == NULL)
                        err("memory failure for %d in squish_lgaps", len-k+1);
                    l = 0;
                    last->start1 = i;
                    last->start2 = j;
                    last->score = 0;
                    have_block = 1;
                }
                /* Update score */
                switch(a->str[k]) {
                    case ALIGN_MATCH:
                        for(m=0; m<N_PROFILE_LETTERS; m++)
                            N[m] = pl->node[i][m] + p2->node[j][m];
                        last->score += SCORE2FLT(log_likelihood(pl->node[i]) -
                            log_likelihood(p2->node[j]));
                }
            }
        }
    }
}

```

```

prml.lg_dirty_change = lg2(dirty_change/3.0);
prml.lg_dirty_del = lg2(dirty_del);
prml.lg_dirty_no_change_del = lg2(1.0-dirty_change /*3.0*/-dirty_del);
prml.lg_dirty_ins = lg2(dirty_ins/4.0);
prml.lg_dirty_no_ins = lg2(1.0-dirty_ins/*4.0*/);

prml.score_pgr = (void**)ll_init();

prml.lgo = lgo;
/* Too small to pass 2^-lgo, which is
   the prob. to open a variant at a basepair */

}

/* Replace functions with macros! */
/* lgap_open must be symmetrical in the weights of the two links, since
   we allow free transitions between the xgap and ygap states.
*/

static score_t lgap_open(float link1, float link2)
{
    /* return -(link1+link2)*prml.lgo/2.0; */
    return FLT2SCORE(-prml.lgo);
}

static score_t lgap_extend(float link1, float link2)
{
    /* return (link1+link2)*prml.lge; */
    return FLT2SCORE(-0);
}
#else
#define lgap_open(x,y) FLT2SCORE(-prml.lgo)
#define lgap_extend(x,y) FLT2SCORE(-0)
#endif

typedef score_t (*score_func)(cprof_node);

static int max_val(cprof p)
{
    int max, i, k;
    max = -1;
    for(i=0; i<p->len; i++)
        for(k=0; k<N_PROFILE_LETTERS; k++)
            if (p->node[i][k] > max)
                max = p->node[i][k];
    return max;
}

static int max_link(cprof p)
{
    int max, i;
    max = -1;
    for(i=0; i<p->len+1; i++)
        if (p->link[i] > max)
            max = p->link[i];
    return max;
}

```

```

break;
case ALIGN_GAP1:
    for(m=0; m<N_PROFILE_LETTERS; m++)
        N[m] = p2->node[j][m];
    N[e_gap] += p1->link[i];
    last->score += SCORE2FUT(log_likelihood(N) -
        log_likelihood(p2->node[j]));
    break;
case ALIGN_GAP2:
    for(m=0; m<N_PROFILE_LETTERS; m++)
        N[m] = p1->node[i][m];
    N[e_gap] += p2->link[j];
    last->score += SCORE2FUT(log_likelihood(N) -
        log_likelihood(p1->node[i]));
    break;
default:
    err("Impossible in squish_lgaps(); '%c'(%d)", a->str[k], a->str[k]);
}

last->str[l++] = a->str[k];
if (a->str[k] != ALIGN_GAP1)
    i++;
if (a->str[k] != ALIGN_GAP2)
    j++;
last->end1 = i;
last->end2 = j;
break;

case ALIGN_LGAP1:
case ALIGN_LGAP2:
    had_block = have_block;
    if (have_block) {
        last->str[l++] = '\0'; /* terminate string */
        last->str = realloc(last->str, l);
        last->next = NULL;
        last->end1--;
        last->end2--;
        have_block = 0;
    }
    if (a->str[k] == ALIGN_LGAP2)
        i++;
    if (a->str[k] == ALIGN_LGAP1)
        j++;
    break;
default:
    err("unknown alignment char '%c' in squish_lgaps", a->str[k]);
    break;
}

if (have_block) {
    last->str[l++] = '\0';
    last->str = realloc(last->str, l);
    last->next = NULL;
    last->end1--;
    last->end2--;
    have_block = 0;
} /* UNUSED */

```

```

free(a->str);
free(a);
return res;
}

static mat_node *cprof_align_build_matrix(cprof p1, cprof p2,
    struct rect_s rect, int mode)
{
    int i, j, k;
    mat_node *mat;
    mat_node *left, *down, *current, *diag, zero, neg_infty;
    int width = rect.i1 - rect.i0 + 1;
    int height = rect.j1 - rect.j0 + 1;
    int max1 = max_val(p1), max2 = max_val(p2);
    int link1 = max_link(p1), link2 = max_link(p2);
    int max_both = (max1 > link1 ? max1 : link1) + (max2 > link2 ? max2 : link2);
    score_func ll_func1 = (max1 > 127) ? log_likelihood : ll_get;
    score_func ll_func2 = (max2 > 127) ? log_likelihood : ll_get;

    for(k=0; k<n_mat_states; k++) {
        zero[k].sc = FLT2SCORE(0.0); /* absolute zero... */
        neg_infty[k].sc = NEG_INFITY; /* borders for global alignment */
    }

    if ((mat = (mat_node *) malloc(width * height * sizeof(mat_node)))
        == NULL)
        err("memory failure in cprof_alignment for %dx%d", width, height);

    /* Create alignment matrix */
    for(i = rect.i0; i <= rect.i1; i++) {
        /* Advance one column and initialise pointers */
        current = mat + (i-rect.i0)*height;
        if (i == rect.i0)
            left = &zero;
        else
            left = mat + (i-1-rect.i0)*height;
        diag = &zero; down = &zero;
        for(j=rect.j0; j <= rect.j1; j++) {
            if (mode & GLOBAL_MODE) {
                if (i == rect.i0)
                    diag = left = &neg_infty;
                if (j == rect.j0)
                    diag = down = &neg_infty;
                if (i == rect.i0 && j == rect.j0)
                    diag = left = down = &zero;
            }

            /* klugy-kode-include file to perform one matrix-update step */
            #define DRAG_ARGS(this,that,score) (this).sc = score;
            #include "ac_update.k"
            #undef DRAG_ARGS

```



```

if (mode & LOCAL_MODE) {
    if ((*current)[e_ll].sc < FLT2SCORE(0))
        (*current)[e_ll].sc = FLT2SCORE(0);
}

/* Advance pointers to next node */
if (i > rect.i0)
    diag = left++;
down = current++;
}

return mat;
}

/* Quadratic space alignment */
cprof_align_cprof_quad_alignment(cprof p1, cprof p2, int mode)
{
    int i, j, k;
    int len;
    mat_node *mat;
    mat_node *current;
    struct rect_s rect;

    score_t best;
    int best_i, best_j;
    enum mat_state best_s;

    struct match_state_s cprof_match(cprof p1, cprof p2,
        mat_node *mat, int mode,
        struct rect_s rect,
        struct match_state_s m);

    cprof_align res;
    struct match_state_s match_data;

    rect.i0 = rect.j0 = 0;
    rect.i1 = p1->len-1;
    rect.j1 = p2->len-1;

    mat = cprof_align_build_matrix(p1, p2, rect, mode & LOCAL_MODE);

    best = FLT2SCORE(-1.0);
    best_i = best_j = -1;

    /* ----- Search for maximal score ----- */
    if (mode & LOCAL_MODE) {
        current = mat;
        for(i=0; i<p1->len; i++)
            for(j=0; j<p2->len; j++) {
                /* This loop will always find a maximum with k == 0 during
                   local alignment. So it is commented out, but included for
                   comparison with overlap alignment. */
                #ifdef USE_SEARCH_LOOP
                    For (k=0; k<n_mat_states; k++)

```

```

if ((*current)[k] > best) {
    best_i = i;
    best_j = j;
    best_s = k;
    best = (*current)[k];
}

} else {
    if ((*current)[e_ll].sc > best) {
        best_i = i;
        best_j = j;
        best_s = e_ll; /* wasted effort */
        best = (*current)[e_ll].sc;
    }

    #endif /* USE_SEARCH_LOOP */
    ++current;
}

} else {
    j = p2->len-1;
    current = mat + j;
    for(i=0; i<p1->len; i++) {
        /* The corresponding loop for local alignment is commented out,
           since local alignments always have terminals at match states */
        for(k=0; k<n_mat_states; k++)
            if ((*current)[k].sc > best) {
                best_i = i;
                best_j = j;
                best_s = k;
                best = (*current)[k].sc;
            }
        current += p2->len;
    }
    i = p1->len-1;
    current = mat + p2->len*i;
    for(j=0; j<p2->len; j++) {
        for(k=0; k<n_mat_states; k++)
            if ((*current)[k].sc > best) {
                best_i = i;
                best_j = j;
                best_s = k;
                best = (*current)[k].sc;
            }
        ++current;
    }
}

/* ----- Backtrack on matrix and get alignment ----- */
len = best_i + best_j + 3;
if ((res = malloc(sizeof(*res))) == NULL)
    err("memory failure for alignment in cprof_quad_alignment");
if ((res->str = malloc(len)) == NULL)
    err("memory failure for alignment string (%d) in cprof_quad_alignment",
        len);

/* pre-terminate string */
res->str[len-1] = '\0';

match_data.p = res->str + len-1;
match_data.i = best_i;
match_data.j = best_j;

```



```

match_data.s = best_s;

match_data = cprof_match(p1, p2, mat, mode, rect, match_data);

memmove(res->str, match_data.p, res->str + len - match_data.p);
res->str = realloc(res->str, strlen(res->str)+1);
res->end1 = best_i;
res->end2 = best_j;
res->start1 = match_data.i;
res->start2 = match_data.j;
res->next = NULL;

/*
printf("Quadratic alignment results: <id,>.<id,> (&d)\n",
res->start1, res->start2, res->end1, res->end2, best_s);
*/

free(mat);
if (mode & SEPARATE_HILITOPS)
    res = squish_lgaps(res, p1, p2);

return res;
}

struct match_state_s cprof_match(cprof p1, cprof p2,
                                mat_node *mat, int mode,
                                struct rect_s rect,
                                struct match_state_s match_data)
{
    int i, j;
    int k;
    int l;
    /* useful loop index */
    /* need to open a new align block? */
    mat_node *current, *left, *down, *diag, zero, neg_infty;
    cprof_node N;
    enum {e_match, e_gap1, e_gap2, e_none} last;
    int width = rect.l1 - rect.i0 + 1;
    int height = rect.j1 - rect.j0 + 1;

    for (l=0; l<n_mat_states; l++) {
        zero[l].sc = FLT2SCORE(0.0);
        neg_infty[l].sc = NEG_INFITY;
    }

    i = match_data.i;
    j = match_data.j;
    last = e_none;

    while (i >= rect.i0 && j >= rect.j0) {
        current = mat + (i-rect.i0)*height + j-rect.j0 ;

```

```

if (i > rect.i0)
    left = mat + (i-1-rect.i0)*height + j-rect.j0 ;
else
    left = (j == rect.j0 || ! (mode & GLOBAL_MODE)) ?
        &zero : &neg_infty;
if (j > rect.j0)
    down = mat + (i-rect.i0)*height + j-1-rect.j0;
else
    down = (i == rect.i0 || ! (mode & GLOBAL_MODE)) ?
        &zero : &neg_infty;
if (i > rect.i0 && j > rect.j0)
    diag = mat + (i-1-rect.i0)*height + j-1-rect.j0;
else
    diag = ((i (mode & GLOBAL_MODE)) ||
            (i == rect.i0 && j == rect.j0)) ?
        &zero : &neg_infty;

/* Local alignments never score negatively */
if ((mode & LOCAL_MODE) && EQ((*current)[e_ll].sc, FLT2SCORE(0.0)))
    break;

switch(match_data.s) {
    #define WRITE_ALIGN(ltr) *(&match_data.p)=(&ltr);

    case e_ll:
        /* Likelihood mode */

        /* match? */
        for (l=0; l<N_PROFILE_LETTERS; l++)
            N[l] = p1->node[i][l] + p2->node[j][l];

        if (EQ((*current)[e_ll].sc - (*diag)[e_ll].sc, log_likelihood(N) -
            log_likelihood(p1->node[i])) -
            log_likelihood(p2->node[j])) {
            WRITE_ALIGN(ALIGN_MATCH);
            i--;
            j--;
            last = e_match;
            break;
        }

        /* gap1? */
        for (l=0; l<N_PROFILE_LETTERS; l++)
            N[l] = p2->node[j][l];
        N[e_gap] += p1->link[i];

        if (EQ((*current)[e_ll].sc - (*down)[e_ll].sc, log_likelihood(N) -
            log_likelihood(p2->node[j])) {
            WRITE_ALIGN(ALIGN_GAP1);
            j--;
            last = e_gap1;
            break;
        }

        /* gap2? */
        for (l=0; l<N_PROFILE_LETTERS; l++)
            N[l] = p1->node[i][l];
        N[e_gap] += p2->link[j];

```

```

if (EQ((*current)[e_ll].sc - (*left)[e_ll].sc, log_likelihood(N) -
    log_likelihood(p1->node(i))) {
    WRITE_ALGN(ALIGN_GAP2);
    i--;
    last = e_gap2;
    break;
}

/* lgap1? */
if (EQ((*current)[e_ll].sc - (*down)[e_lgap1].sc,
    lgap_open(p1->link[i], p2->link[j])) {
    /* printf("Leave lgap1 <td %d>\n", i, j); */
    WRITE_ALGN(ALIGN_LGAP1);
    j--;
    match_data.s = e_lgap1;
    break;
}

/* lgap2? */
if (EQ((*current)[e_ll].sc - (*left)[e_lgap2].sc,
    lgap_open(p2->link[j], p1->link[i])) {
    /* printf("Leave lgap2 <td %d>\n", i, j); */
    WRITE_ALGN(ALIGN_LGAP2);
    i--;
    match_data.s = e_lgap2;
    break;
}

/* Impossible! Where did we come from? */
err("error in backtracking from ll state, <td %d>", i, j);
break;
/* UNREACHED */

case e_lgap1:
    /* (long) X-gap mode */

    /* match? */
    if (EQ((*current)[e_lgap1].sc - (*down)[e_lgap1].sc,
        lgap_extend(p1->link[i], p2->link[j])) {
        WRITE_ALGN(ALIGN_LGAP1);
        j--;
        match_data.s = e_lgap1;
        break;
    }

    /* if (EQ((*current)[e_lgap1].sc - (*down)[e_lgap2].sc,
        lgap_extend(p2->link[j], p1->link[i])) {
        j--;
        match_data.s = e_lgap2;
        break;
    }

    /* if (EQ((*current)[e_lgap1].sc - (*current)[e_ll].sc,
        lgap_open(p1->link[i], p2->link[j])) {
        /* printf("Enter lgap1 <td %d>\n", i, j); */
        match_data.s = e_ll;
        break;
    }

    /* Impossible! Where did we come from? */
    err("error in backtracking from lgap1 state, <td %d>", i, j);
    break;
/* UNREACHED */

```

align_cprof.c

```

case e_lgap2:
    /* (long) Y-gap mode */

    if (EQ((*current)[e_lgap2].sc - (*left)[e_lgap2].sc,
        lgap_extend(p2->link[j], p1->link[i])) {
        WRITE_ALGN(ALIGN_LGAP2);
        i--;
        match_data.s = e_lgap2;
        break;
    }

    if (EQ((*current)[e_lgap2].sc - (*current)[e_lgap1].sc,
        lgap_extend(p1->link[i], p2->link[j])) {
        /* printf("Enter lgap2 <td %d>\n", i, j); */
        match_data.s = e_lgap1;
        break;
    }

    if (EQ((*current)[e_lgap2].sc - (*current)[e_ll].sc,
        lgap_open(p2->link[j], p1->link[i])) {
        /* printf("Enter lgap2 <td %d>\n", i, j); */
        match_data.s = e_ll;
        break;
    }

    /* Impossible! Where did we come from? */
    err("error in backtracking from lgap2 state, <td %d>", i, j);
    break;
/* UNREACHED */

default:
    err("unknown state %d in backtracking, <td %d>", match_data.s, i, j);
}

#undef WRITE_ALGN

if ((match_data.s == e_ll && (last == e_match || last == e_gap2)) ||
    match_data.s == e_lgap2)
    i++;
if ((match_data.s == e_ll && (last == e_match || last == e_gap1)) ||
    match_data.s == e_lgap1)
    j++;

match_data.i = i;
match_data.j = j;

return match_data;
}

/* Update endpoints (alignment ends here) */

/* ===== Linear-space alignment routines ===== */

/* Find alignment record in linear space */
static struct alignment_rec_s
cprof_linear_alignment(cprof pl, cprof p2, int mode)
{
    int i, j, k;
    lmat_node *this_col, *prev_col, *swap_tmp;
    lmat_node *left, *down, *current, *diag, l_zero, d_zero, dd_zero;

```

align_cprof.c

```

score_t best;
struct alignment_rec_s res;

int max1 = max_val(p1), max2 = max_val(p2);
int link1 = max_link(p1), link2 = max_link(p2);
int max_both = (max1 > link1 ? max1 : link1) + (max2 > link2 ? max2 : link2);
score_func ll_func = (max1 > 127) ? log_likelihood : ll_get;
score_func ll_func1 = (max1 > 127) ? log_likelihood : ll_get;
score_func ll_func2 = (max2 > 127) ? log_likelihood : ll_get;

for(k=0; k<n_mat_states; k++) {
    l_zero[k].sc = FLT2SCORE(0.0); /* absolute zero... */
    dd_zero[k].sc = FLT2SCORE(0.0);
    d_zero[k].sc = FLT2SCORE(0.0);
}

if ((this_col = (lmat_node *) malloc(p2->len*sizeof(lmat_node)))
    == NULL)
    err("memory failure (1) in cprof_linear_alignment for %d", p2->len);
if ((prev_col = (lmat_node *) malloc(p2->len*sizeof(lmat_node)))
    == NULL)
    err("memory failure (2) in cprof_linear_alignment for %d", p2->len);
best = FLT2SCORE(-1.0);

/* Scan alignment matrix */
for(i=0; i<p1->len; i++) {
    /* Advance one column and initialise pointers */
    swap_tmp = this_col;
    this_col = prev_col;
    prev_col = swap_tmp;

    current = this_col;
    if (i == 0) {
        left = &l_zero;
        l_zero[e_ll].i0 = l_zero[e_lgap1].i0 = l_zero[e_lgap2].i0 = 0;
    }
    else
        left = prev_col;
    down = &d_zero;
    diag = &dd_zero;
    dd_zero[e_ll].i0 = dd_zero[e_lgap1].i0 = dd_zero[e_lgap2].i0 = i;
    dd_zero[e_ll].j0 = dd_zero[e_lgap1].j0 = dd_zero[e_lgap2].j0 = 0;
    d_zero[e_ll].i0 = d_zero[e_lgap1].i0 = d_zero[e_lgap2].i0 = i;
    d_zero[e_ll].j0 = d_zero[e_lgap1].j0 = d_zero[e_lgap2].j0 = 0;

    for(j=0; j < p2->len; j++) {
        if (i == 0) {
            dd_zero[e_ll].j0 = dd_zero[e_lgap1].j0 = dd_zero[e_lgap2].j0 =
                l_zero[e_ll].j0 = l_zero[e_lgap1].j0 = l_zero[e_lgap2].j0 = j;
            dd_zero[e_ll].i0 = dd_zero[e_lgap1].i0 = dd_zero[e_lgap2].i0 =
                l_zero[e_ll].i0 = l_zero[e_lgap1].i0 = l_zero[e_lgap2].i0 = 0;
        }

        /* ----- Update matrix elements ----- */

```

```

/* klugy-kode-include file to perform one matrix-update step */
#define DRAG_ARGS(this,that,score)
{
    (this).sc = score;
    (this).i0 = (that).i0;
    (this).j0 = (that).j0;
}

#include "ac_update.k"

#undef DRAG_ARGS

/* ----- Search for maximal score ----- */
if (mode & LOCAL_MODE) {
    if ((*current)[e_ll].sc <= FLT2SCORE(0)) {
        (*current)[e_ll].sc = FLT2SCORE(0);
        (*current)[e_ll].i0 = i;
        (*current)[e_ll].j0 = j;
    }

    if ((*current)[e_ll].sc > best) {
        best = (*current)[e_ll].sc;
        res.i1 = i;
        res.j1 = j;
        res.st = e_ll;
        res.i0 = (*current)[e_ll].i0;
        res.j0 = (*current)[e_ll].j0;
    }

    else if (i == p1->len-1 || j == p2->len-1) {
        for(k=0; k<n_mat_states; k++)
            if ((*current)[k].sc > best) {
                best = (*current)[k].sc;
                res.i1 = i;
                res.j1 = j;
                res.st = k;
                res.i0 = (*current)[e_ll].i0;
                res.j0 = (*current)[e_ll].j0;
            }
    }

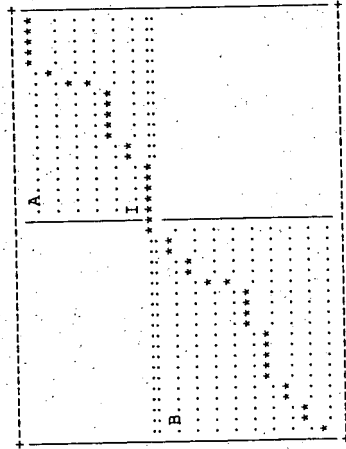
    /* ----- Advance pointers to next node ----- */
    if (i > 0)
        diag = left++;
    down = current++;
}

/*
    printf("Linear alignment results: <%d,%d>...<%d,%d> (%d)\n",
        res.i0, res.j0, res.i1, res.j1, res.st);
*/

```

```
return res;
}
```

```
/*
** Consider a path as described below. Denote the intersection with the
** halfway mark by I (we may find it in quadratic time and linear space).
** Then the complete alignment may be found (recursively) by
** concatenating the complete alignments in the 2 rectangles through
** which the alignment passes. We do this, until we reach rectangles for
** which quadratic-space alignment isn't prohibitive.
*/
```



```
/*
** In practice, it is more convenient to take the rectangles in reverse
** order, since that way we have the state calculated by the previous
** stage, and don't need to store it anywhere.
*/
```

```
/*
** cprof_linear_split_rect --
**
** Split rectangle bounding alignment into 2 rectangles, each of half
** the original's width, such that the alignment passes through their
** shared corner. See the diagram above.
**
** Uses linear space. Memory (at least sizeof(llmat_node) * height_of_
** _rectangle) should be allocated and freed by topmost calling
** routine, to avoid having to do so at each node of the recursion
** (this would cause O(width_of_upmost_rectangle) allocations, which
** is rather bad...).
**
static void *column1 = NULL, *column2 = NULL;

static void
cprof_linear_split_rect(cprof p1, cprof p2, /* profiles */
struct rect_s rect, /* alignment bounds */
```

align_cprof.c

```
enum mat_state state,
struct rect_s *A,
struct rect_s *B)
```

```
{
    int i, j, k;
    llmat_node *this_col, *prev_col, *swap_tmp;
    llmat_node *left, *down, *current, *diag, zero, neg_infty;
    int width = rect.i1-rect.i0+1;
    int height = rect.j1-rect.j0+1;

    int max1 = max_val(p1), max2 = max_val(p2);
    int link1 = max_link(p1), link2 = max_link(p2);
    int max_both = (max1 > link1 ? link1 : link1) + (max2 > link2 ? link2);
    score_func ll_func = (max1 > 127) ? log_likelihood : ll_get;
    score_func ll_func2 = (max1 > 127) ? log_likelihood : ll_get;
    score_func ll_func2 = (max2 > 127) ? log_likelihood : ll_get;

    for(k=0; k<n_mat_states; k++) {
        zero[k].sc = FLT2SCORE(0.0);
        neg_infty[k].sc = NEG_INFITY;
        zero[k].jj = -1;
        neg_infty[k].jj = -1;
    }
    /* Won't be propagated past halfway */

    this_col = column1;
    prev_col = column2;

    if 0 {
        if ((this_col = (llmat_node *) malloc(height*sizeof(llmat_node)))
            == NULL)
            err("memory failure (1) in cprof_linear_split_rect for %d", height);
        if ((prev_col = (llmat_node *) malloc(height*sizeof(llmat_node)))
            == NULL)
            err("memory failure (2) in cprof_linear_split_rect for %d", height);
        #endif

        /* Scan alignment matrix */
        for(i=rect.i0; i<=rect.i1; i++) {
            /* Advance one column and initialise pointers */
            swap_tmp = this_col;
            this_col = prev_col;
            prev_col = swap_tmp;
            current = this_col;

            left = prev_col;

            for(j=rect.j0; j<=rect.j1; j++) {
                if (i == rect.i0)
                    diag = left = &neg_infty;
                if (j == rect.j0)
                    diag = down = &neg_infty;
                if (i == rect.i0 && j == rect.j0)
                    diag = left = down = &zero;

                /* klugy-kode-include file to perform one matrix-update step */
            }
        }
    }
}
```

align_cprof.c

```

#define DRAG_ARGS(this, that, score)
{
    (this).sc = score;
    (this).jj = (that).jj;
}

#include "ac_update.k"

#undef DRAG_ARGS

/* ----- Don't need to search for maximal score ----- */

/* ----- Advance pointers to next node ----- */
if (i > rect.i0)
    diag = left++;
down = current++;
}

if (i == rect.i0+width/2)
/* Initialise variables for halfway crossing */
for (j=rect.j0; j<rect.j1; j++)
    this_col[j-rect.j0][e.ll].jj = this_col[j-rect.j0][e_lgapl].jj =
        this_col[j-rect.j0][e_lgap2l].jj = j;
}

/* Routine's output */
A->i1 = rect.i1;
A->j1 = rect.j1;
A->i0 = B->i1 = rect.i0 + width/2;
A->j0 = B->j1 = this_col[height-1][state].jj;
B->i0 = rect.i0;
B->j0 = rect.j0;

if 0
free(this_col);
free(prev_col);
#endif
)

```

```

/*
 * Given a rectangle containing the alignment (and the final state),
 * compute (part of) the alignment string. This involves a direct call
 * to cprof_align_build_matrix and then cprof_match if quadratic space
 * can be used, otherwise a cprof_linear_split_rect to split into 3
 * rectangles, and a recursive call into each of them.
 *
 * Returns the final state (needed for the recursive calls to work).
 */

static struct match_state_s cprof_linear_match(cprof pl, cprof p2,
struct rect_s rect,

```

align_cprof.c

```

struct match_state_s match_data)
{
    if ((long)(rect.i1+1-rect.i0) * (rect.j1+1-rect.j0) * sizeof(mat_node)
        <= cprof_align_max_mem) {
        /* Calculate directly, in quadratic space */

        mat_node *mat;

        /* Build global-type alignment matrix (will always work by the time
         we're here! */
        mat = cprof_align_build_matrix(pl, p2, rect, GLOBAL_MODE);
        match_data = cprof_match(pl, p2, mat, GLOBAL_MODE,
                                rect, match_data);

        free(mat);
        return match_data;
    }
    else {
        /* Calculate recursively, in linear space */
        struct rect_s A, B;

        cprof_linear_split_rect(pl, p2, rect, match_data.s, &A, &B);

        #if 0
        printf("<td &td &td> ==> <td &td &td> <td &td &td>\n",
            rect.i0, rect.j0, rect.i1, rect.j1,
            A.i0, A.j0, A.i1, A.j1, B.i0, B.j0, B.i1, B.j1);
        #endif

        match_data = cprof_linear_match(pl, p2, A, match_data);
        assert(match_data.i == A.i0 && match_data.j == A.j0);

        /* Do overlap letter of match again */
        match_data.p++;
        match_data = cprof_linear_match(pl, p2, B, match_data);
        assert(match_data.i == rect.i0 && match_data.j == rect.j0);
        return match_data;
    }
}

/* * Driver routine for quadratic or linear space matching
 */

cprof_align cprof_alignment(cprof pl, cprof p2, int mode)
{
    struct rect_s rect;
    cprof_align res;
    struct match_state_s match_data;
    struct alignment_rec_s align;
    int len;
    int i, j;

    /* Allocate costs arrays */
    if ((cost1 = (score_t *) malloc(pl->len * sizeof(score_t))) == NULL ||
        (cost2 = (score_t *) malloc(p2->len * sizeof(score_t))) == NULL) {
        if (cost1)

```

align_cprof.c

```

    free(cost1);
    err("memory failure in cprof_alignment for %d or %d", p1->len, p2->len);
}

/* Update scores for both profiles */
for(i=0; i<p1->len; i++)
    cost1[i] = log_likelihood(p1->node[i]);
for(j=0; j<p2->len; j++)
    cost2[j] = log_likelihood(p2->node[j]);

if (p1->len * p2->len * sizeof(mat_node) <= cprof_align_max_mem) {
    /* Run quadratic-space routines */
    res = cprof_quad_align(p1, p2, mode);
    free(cost1);
    free(cost2);
    return res;
    /* UNREACHED */
}

/* Set up linear-space alignment */
if ((column1 = malloc(p2->len * sizeof(llmat_node))) == NULL ||
    (column2 = malloc(p2->len * sizeof(llmat_node))) == NULL) {
    if (column1)
        free(column1);
    err("memory failure in cprof_alignment for %d", p2->len);
}

/* Find rectangle where optimum lives */
align = cprof_linear_align(p1, p2, mode);

/* Prepare rectangle and alignment string writers */
rect.i0 = align.i0;
rect.j0 = align.j0;
rect.i1 = align.i1;
rect.j1 = align.j1;

len = (rect.i1-rect.i0 + 1) * (rect.j1-rect.j0 + 1) + 3;
if ((res = malloc(sizeof(*res))) == NULL)
    err("memory failure for alignment in cprof_alignment");
if ((res->str = malloc(len)) == NULL)
    err("memory failure for alignment string (%d) in cprof_alignment",
        len);

/* pre-terminate string */
res->str[len-1] = '\0';
match_data.p = res->str + len-1;
match_data.i = rect.i1;
match_data.j = rect.j1;
match_data.s = align.st;

match_data = cprof_linear_match(p1, p2, rect, match_data);

memmove(res->str, match_data.p, res->str + len - match_data.p);
res->str = realloc(res->str, strlen(res->str)+1);
res->end1 = rect.i1;
res->end2 = rect.j1;
res->start1 = match_data.i;
res->start2 = match_data.j;

```

align_cprof.c

```

free(column1);
free(column2);
if (mode & SEPARATE_HILLTOPS)
    res = squish_gaps(res,p1,p2);
free(cost1);
free(cost2);
return res;
}

/* Use maximal likelihood to compute a string corresponding to a cprof.
 */

/* Compute the letter corresponding to a single cprof location (or x
 * for a gap). Ambiguous letters are only returned if the maximal
 * likelihood occurs for several letters. In particular, these
 * letters must appear the same number of times in the cprof.
 */

/* KLUGE to use realnode to get "better" results. Work out how many N's
 * there were here, and distribute them equally among the letters. Since
 * everything's an int, we multiple everything by N_PROFILE_LETTERS-1,
 * which should still be OK (all is linear).
 */
static char amb[17] = "XACMGSRVTWYHKDEN";

char cprof_node_char(cprof p, int loc,
                    int *is_error, int *is_corrected)
{
    int clean_N(N_PROFILE_LETTERS+1);
    int N(N_PROFILE_LETTERS+1);
    int dirty_N(N_PROFILE_LETTERS+1);
    int clean_sum, sum, dirty_sum;
    int *Nptr;
    int i;
    unsigned char ch;

    float score, best_score;
    int best_mask, best_multiplicity;

    int first;
    if (is_error)
        *is_error = 0;
    if (is_corrected)
        *is_corrected = 0;

    /* Zero counts */
    for(i=0; i<N_PROFILE_LETTERS+1; i++)
        clean_N[i] = N[i] = dirty_N[i] = 0;

    /* Count letters */

```

align_cprof.c

```

for(i=0; i<p->width; i++)
    if (! CPROF_IS_BLANK(ch = *cprof_seq_ptr(p, loc, i))) {
        Nptr = CPROF_IS_DIRTY(ch) ? dirty_N :
        p->very_clean[i] ? clean_N : N;
        if (CPROF_IS_N(ch)) /* Count one N */
            Nptr[N_PROFILE_LETTERS]++;
        else /* Count one letter */
            Nptr[CPROF_LETTER_NO(ch)]++;
    }

clean_sum = sum = dirty_sum = 0;
for(first = -1, i=1; i<N_PROFILE_LETTERS+1; i++) {
    if (i < N_PROFILE_LETTERS && (N[i] || dirty_N[i]) && first == -1)
        first = i;
    clean_sum += clean_N[i];
    sum += N[i];
    dirty_sum += dirty_N[i];
}

/* Flag "error" if we didn't have total consensus */
if (is_error &&
    first >= 0 &&
    clean_N[first] + N[first] + dirty_N[first] < clean_sum + sum + dirty_sum
    *is_error = 1;

/* Case A: Calculate score for a gap, with some (mistaken) insertions */
best_score =
    N[e_gap] * prm.lg_no_ins +
    sum * prm.lg_ins +
    clean_N[e_gap] * prm.lg_clean_no_ins +
    clean_sum * prm.lg_clean_ins +
    dirty_N[e_gap] * prm.lg_dirty_no_ins +
    dirty_sum * prm.lg_dirty_ins;
best_mask = 0;
best_multiplicity = 1;

/* Case B: Calculate score for i'th letter, with some (mistaken) changes,
and also some (mistaken) deletions (these appear as gaps) */
for(i=1; i<N_PROFILE_LETTERS; i++) {
    score = -2.0 +
        (N[i] + (float) N[N_PROFILE_LETTERS] / 4.0) * prm.lg_no_change_del +
        (sum - N[i] - (float) N[N_PROFILE_LETTERS] / 4.0) * prm.lg_change +
        N[e_gap] * prm.lg_del +
        (clean_N[i] + (float) clean_N[N_PROFILE_LETTERS] / 4.0) * prm.lg_clean_no_
        change_del +
        (clean_sum - clean_N[i] - (float) clean_N[N_PROFILE_LETTERS] / 4.0) * prm.
        lg_clean_change +
        clean_N[e_gap] * prm.lg_clean_del +
        (dirty_N[i] + (float) dirty_N[N_PROFILE_LETTERS] / 4.0) * prm.lg_dirty_no_
        change_del +
        (dirty_sum - dirty_N[i] - (float) dirty_N[N_PROFILE_LETTERS] / 4.0) * prm.
        lg_dirty_change +
        dirty_N[e_gap] * prm.lg_dirty_del;
    if (score > best_score) {
        best_score = score;
        best_mask = 1 << (i-1);
        best_multiplicity = 1;
    }
}
else if (score == best_score) {
    best_mask |= 1 << (i-1);
    best_multiplicity++;
}

```

align_cprof.c

```

    }
    if (is_corrected && is_error && *is_error && best_score > lg2(.95))
        *is_corrected = 1;
    return amb[best_mask];
}

char *cprof_compute_assembly(cprof p,
    int ignore_gaps, int *probable_mistakes,
    int *corrected)
{
    int i, is_error, is_corrected, j;
    char *assembly, ch;
    char *res;

    if ((assembly = (char *) malloc(p->len+1)) == NULL)
        err("cprof_compute_assembly: memory failure");
    /* trying to allocate &d for 'assembly', p->len+1);

    if (probable_mistakes)
        *probable_mistakes = 0;
    if (corrected)
        *corrected = 0;
    for(i=j=0; i < p->len; i++) {
        ch = cprof_node_char(p, i, &is_error, &is_corrected);
        if (is_error && probable_mistakes) (*probable_mistakes)++;
        if (is_corrected && corrected) (*corrected)++;
        if (ch != 'X' || ! ignore_gaps)
            assembly[j++] = ch;
    }
    assembly[j++] = '\0';
    res = strdup(assembly);
    free(assembly);
    return res;
}

int cprof_num_print(FILE *fp, char *prefix, cprof p, int inc_cnt, int exc_cnt)
{
    int i;
    char ch;

#define SUM(A) ((A)[e_a] + (A)[e_c] + (A)[e_g] + (A)[e_t])
    for(i=0; i<p->len; i++) {
        if (inc_cnt & 10 == 0)
            fprintf(fp, "%sCons A C G T N gap Loc: %d (Excluding 'X': %d)\n",
                prefix, inc_cnt, exc_cnt);
        ch = cprof_node_char(p, i, NULL, NULL);
        fprintf(fp, "%s %3c%3d%3d%3d%3d%3d%3d\n",
            prefix, ch,
            p->realnode[i][e_a], p->realnode[i][e_c], p->realnode[i][e_g],
            p->realnode[i][e_t],
            SUM(p->node[i]) - SUM(p->realnode[i]),

```

align_cprof.c

A-16

Sun Aug 9 10:33:18 1998

Listing for Adam Sartei

```
(
    int i1, i2, i;
    int l;
    char chl, ch2;
    int id, sim;

    l = 0;
    id = sim = 0;
    for(i=0; i1=start1, i2=start2; str[i]; i++)
        switch(str[i]) {

            case ALIGN_MATCH:
                chl = cprof_node_char(p1, i1, NULL, NULL);
                ch2 = cprof_node_char(p2, i2, NULL, NULL);
                if (identical_letters(chl, ch2) || (chl == 'X' && ch2 == 'X'))
                    ++id;
                if (similar_letters(chl, ch2))
                    ++sim;
                i1++; i2++;
                l++;
                break;

            case ALIGN_GAP1:
                ch2 = cprof_node_char(p2, i2, NULL, NULL);
                if (ch2 != 'X')
                    l++;
                i2++;
                break;

            case ALIGN_GAP2:
                chl = cprof_node_char(p1, i1, NULL, NULL);
                if (chl != 'X')
                    l++;
                i1++;
                break;

            default:
                err("Unknown alignment character '%c' (%d) in cprof_calc_id_sim()",
                    str[i], str[i]);
                break;
        }
        /* UNREACHED */
    }

    *id_percent = (double) 100.0 * id / l;
    *sim_percent = (double) 100.0 * sim / l;
}

/* Experimental alignment improver
 *
 * The goal is to improve the quality of cprofs, using the
 * multiple-alignment information. This in an effort to get around the
 * worst effects of the pairwise-alignment technique. "Quality" is still
 * measured using the maximal-likelihood formulae, so it will "not"
 * overcome limitations of maximal likelihood.
 *
 * Improvement works by attempting to improve small windows (WINDOW_SIZE
 * in length, typically <10) of the alignment, by shifting the gaps
 * around (in particular, the number of gaps will never be changed, which
```

align_cprof.c

Sun Aug 9 10:33:18 1998

Listing for Adam Sartei

```
        p->node[i][e_gap]);
    if (ch != 'X')
        ++exc_cnt;
    ++inc_cnt;
}
return exc_cnt;
}

void cprof_print(cprof p)
{
    (void) cprof_num_print(stdout, "", p, 0, 0);
}

/* Calculate identity and similarity percentages for an alignment of 2
 * profiles.
 *
 * _Identity_ is defined as having exactly the same letter of A,C,G,T at
 * 2 matched locations, or having an X matched against an X or a gap.
 *
 * _Similarity_ means the above, or having 2 "compatible" letters
 * (e.g. an A or C against an M, or an M and an M). An X must still
 * match an X or a gap to be counted, since it denotes the 'absence' of
 * any letter in the maximal-likelihood interpretation of the profile.
 *
 * */
static int identical_letters(char chl, char ch2)
{
    static char id_letters[] = "XACGT";
    char *p1 = strchr(id_letters, chl);
    char *p2 = strchr(id_letters, ch2);
    return (p1 /* && p2 */ && (p1 == p2));
}

static int similar_letters(char chl, char ch2)
{
    char *p1 = strchr(amb, chl);
    char *p2 = strchr(amb, ch2);
    int i1, i2;

    if ((i1 || i2) || (! p2))
        err("Internal error: impossible letters '%c' (%d)/'%c' (%d)",
            chl, chl, ch2, ch2);
    i1 = p1-amb;
    i2 = p2-amb;
    if (chl == 'X' || ch2 == 'X')
        return chl == ch2;
    return ((i1 | i2) == i1) || ((i1 | i2) == i2);
}

void cprof_calc_id_sim(cprof p1, cprof p2,
    int start1, int start2, char *str,
    double *id_percent, double *sim_percent)
```

align_cprof.c

Sun Aug 9 10:33:18 1998

Using for Adam Sartiell

```

* is obviously sub-optimal). In this initial version, we attempt to
* improve the cprof one strand at a time (so it is pointless to use it
* for width <=2).
* Code is written for experimentation, not for speed.
*/

#define WINDOW_SIZE 9

/* Advance to the next combination
* pos represents an array of the sz gap positions (thus there are
* WINDOW_SIZE-sz letters).
* Return 1 if there actually was a next combination, or 0 if we're done.
*/
short advance(int *pos, int sz)
{
    int i;
    int limit;

    if (sz == 0)
        return 0;

    i = sz-1;
    limit = WINDOW_SIZE-1;
    do {
        if (pos[i] < limit) {
            pos[i]++;
            while (++i < sz)
                pos[i] = pos[i-1]+1;
            return 1;
        }
        limit = pos[i]-1;
        --i;
    } while (i >= 0);

    return 0;
}

/* Represent a combination
* Say that the original string was AA-CT-GG--. Each combination
* represents some (other) ordering of the gaps inside the string
* AACTGG. Given the desired positions for the gaps, writes into the
* output string the string AACTGG, separated by the gaps at the
* appropriate locations.
*/
void represent(int *pos, int sz, unsigned char *orig, unsigned char *res)
{
    int o_idx;
    int g_idx;
    int r_idx;
    int p_idx;

    for (r_idx = g_idx = o_idx = p_idx = 0; r_idx < WINDOW_SIZE; r_idx++)

```

align_cprof.c

Sun Aug 9 10:33:18 1998

Using for Adam Sartiell

```

if (p_idx < sz && r_idx == pos[p_idx]) {
    while (! CPROF_IS_BLANK(orig[g_idx]) &&
           CPROF_LETTER_NO(orig[g_idx]) > 0)
        ++g_idx;
    res[r_idx] = orig[g_idx++];
    p_idx++;
}
else {
    while (! CPROF_IS_BLANK(orig[o_idx]) &&
           CPROF_LETTER_NO(orig[o_idx]) == 0)
        ++o_idx;
    res[r_idx] = orig[o_idx++];
}
}

/* rect_score -- Get the score represented by a "rectangle" of
* characters, assuming window NEW has been substituted for ORIG in
* one of the strands.
*/
static score_t rect_score(cprof p, int loc,
                          unsigned char *orig, unsigned char *new)
{
    cprof_node N;
    int i, k;
    score_t sc;

    sc = 0;
    for (i=0; i<WINDOW_SIZE; i++) {
        for (k=0; k<N_PROFILE_LETTERS; k++)
            N[k] = p->nnode[loc+i][k];
        if (! CPROF_IS_BLANK(orig[i]))
            --N[CPROF_LETTER_NO(orig[i])];
        if (! CPROF_IS_BLANK(orig[i]))
            ++N[CPROF_LETTER_NO(orig[i])];
        sc += log_likelihood(N);
    }
    return sc;
}

/* Return a (static) pointer to a printable representation of a window.
*/
static char *printable(unsigned char str[WINDOW_SIZE])
{
    static char res[WINDOW_SIZE];
    int i;

    for (i=0; i<WINDOW_SIZE; i++)
        res[i] = CPROF_IS_BLANK(str[i]) ? ' ' : bases[CPROF_LETTER_NO(str[i])];
    return res;
}

/* try_improve -- Try to improve a window in the profile.
*/

```

align_cprof.c

```

* Returns 0 or 1
*/
static int try_improve(cprof p, int row, int loc)
{
    score_t sc;
    int pos[WINDOW_SIZE];
    unsigned char best[WINDOW_SIZE], orig[WINDOW_SIZE], new[WINDOW_SIZE];
    score_t best_sc;
    int ngaps;
    int i, k;
    int improved, improved_here;
    unsigned char ch;

    improved = 0;
    for(i=0; best_sc = 0; i<WINDOW_SIZE; i++)
        best_sc += log_likelihood(p->node[i+loc]);

    improved_here = 0;
    ngaps = 0;
    for(i=0; i<WINDOW_SIZE; i++) {
        ch = orig[i] = best[i] = *cprof_seq_ptr(p, i+loc, row);
        if (CPROF_IS_BLANK(ch)) /* have spaces -- don't permute */
            return 0;
        else if (CPROF_LETTER_NO(ch) == 0)
            ngaps++;
    }

    for(k=0; k<ngaps; k++) /* Initialise gap locations */
        pos[k] = k;

    do {
        /* Permute gaps among letters */
        represent(pos, ngaps, orig, new);
        sc = rect_score(p, loc, orig, new);
        if (sc > best_sc) { /* improvement? */
            #ifdef SHOW_IMPROVE
                /* shallow magic -- see comp.lang.c FAQ, Q11.17 and Q11.18 */
                /* These are the contortions we need to go through to quote
                macros into strings. When WINDOW_SIZE is 9, the format string
                looks like " %9s %9s\n" (but with more spaces).
                */
                #define STR(x) #x
                #define XSTR(x) STR(x)
                printf(" %f --> %f\n", SCORE2FLT(best_sc), SCORE2FLT(sc));
                for(i=0; i<p->width; i++)
                    printf(" %c %." XSTR(WINDOW_SIZE) "s %c\n",
                        "s %." XSTR(WINDOW_SIZE) "s %c\n",
                        i == row ? '>' : ' ',
                        i == row ? printable(orig) : printable(new),
                        printable(cprof_seq_ptr(p, loc, i)),
                        i == row ? '<' : ' ');
            #endif
            #undef STR
            #undef XSTR
            #endif /* SHOW_IMPROVE */
            best_sc = sc;
            for(i=0; i<WINDOW_SIZE; i++)
                best[i] = new[i];
            improved_here = 1;
            improved = i;
        }
    }
}

```

align_cprof.c

```

) while (advance(pos, ngaps)); /* next combination of gaps/letters */

if (improved_here) { /* incorporate improvement into strand */
    for(i=0; i<WINDOW_SIZE; i++) {
        *cprof_seq_ptr(p, loc+i, row) = best[i];
        cprof_count(p, loc+i, p->node[loc+i], p->realnode[loc+i]);
    }
} else /* restore original */
    for(i=0; i<WINDOW_SIZE; i++)
        *cprof_seq_ptr(p, loc+i, row) = orig[i];

return improved;
}

/*
 * incorporate_seqs -- incorporate changes from sequences into counts
 *
 * This provides a translation from work done at the window starting at
 * pos back to properly counted profiles.
 */
static void incorporate_seqs(cprof p, int pos)
{
    int i;
    for(i=0; i<WINDOW_SIZE; i++)
        cprof_count(p, i+pos, p->node[i+pos], p->realnode[i+pos]);
}

/* Get # of gaps before AXIS in a window around it with given OFFSET */
static int first_gap(cprof p, int row, int axis, int offset)
{
    int i, n;
    unsigned char ch;
    for(i=0, n=0; i<offset; i++) {
        ch = *cprof_seq_ptr(p, axis-offset+i, row);
        if (! CPROF_IS_BLANK(ch) && CPROF_LETTER_NO(ch) == 0)
            n++;
    }
    return n;
}

static void update_gap_loc(int **gap_loc, cprof p, int row,
                           int loc, int first_gap_num)
{
    int i, n;
    unsigned char ch;
    for(i=0, n=0; i<WINDOW_SIZE; i++) {
        ch = *cprof_seq_ptr(p, loc+i, row);
        if (! CPROF_IS_BLANK(ch) && CPROF_LETTER_NO(ch) == 0) {
            gap_loc[row][first_gap_num+n] = loc+i;
            n++;
        }
    }
}

```

align_cprof.c

```

int improve_cprof(cprof p)
{
    int i, j, k, loc, m, n;
    int num_times;
    int improved, did_something = 0;

    int *n_gaps;
    int **gap_loc;

    unsigned char ch;

    struct shuffle_loc_s {
        int row, gap, win;
    } *shuffle = NULL;

    if (p->len <= WINDOW_SIZE) return 0; /* not worth the effort (??) */

    /* Allocate memory */
    if ((n_gaps = malloc(p->width * sizeof(int))) == NULL ||
        (gap_loc = malloc(p->width * sizeof(int *))) == NULL) {
        err("memory failure in improve_cprof for gaps in row %d",
            p->len);
        return 0;
    }
    for(j=0; j<p->len; j++) {
        ch = *cprof_seq_ptr(p, j, i);
        if (! CPROF_IS_BLANK(ch) && CPROF_LETTER_NO(ch) == 0)
            ++n_gaps[j];
    }
    if (n_gaps[i] &&
        (gap_loc[i] = (int *) malloc(n_gaps[i] * sizeof(int))) == NULL)
        err("memory failure in improve_cprof for %d for gaps in row %d",
            n_gaps[i], i);
    n += n_gaps[i];

    if (n) {
        if ((shuffle = malloc(n * WINDOW_SIZE *
            sizeof(struct shuffle_loc_s))) == NULL)
            err("memory failure in improve_cprof for %d gapwindows", n);

        /* Load gap_loc */
        for(i=0; i<p->width; i++) {
            for(j=0; loc=0; j<p->len; j++) {
                ch = *cprof_seq_ptr(p, j, i);
                if (! CPROF_IS_BLANK(ch) && CPROF_LETTER_NO(ch) == 0)
                    gap_loc[i][loc++] = j;
            }
            assert(loc == n_gaps[i]);
        }

        num_times = 0;
        do {
            /* Don't go through loop too many times! */
            /* Randomise order */

```

```

for(i=0, m=0; i<p->width; i++)
    for(j=0; j<n_gaps[i]; j++)
        for(k=0; k<WINDOW_SIZE; k++) {
            shuffle[m].row = i;
            shuffle[m].gap = j;
            shuffle[m].win = k;
            ++m;
        }
    assert(m == n * WINDOW_SIZE);

    for(m=0; m<n; m++) {
        struct shuffle_loc_s tmp;
        int r = m + rand() % (n-m);
        tmp = shuffle[r];
        shuffle[r] = shuffle[m];
        shuffle[m] = tmp;
    }

    /* Try to improve something */
    improved = 0;
    for(m=0; m<n; m++) {
        int win_start = gap_loc[shuffle[m].row][shuffle[m].gap] -
            shuffle[m].win;
        int first;

        /* Try to improve alignment */
        if (! (0 <= win_start && win_start+WINDOW_SIZE <= p->len))
            continue;

        first = gap_loc[shuffle[m].row][shuffle[m].gap] -
            first_gap[p, shuffle[m].row,
                gap_loc[shuffle[m].row][shuffle[m].gap],
                shuffle[m].win];

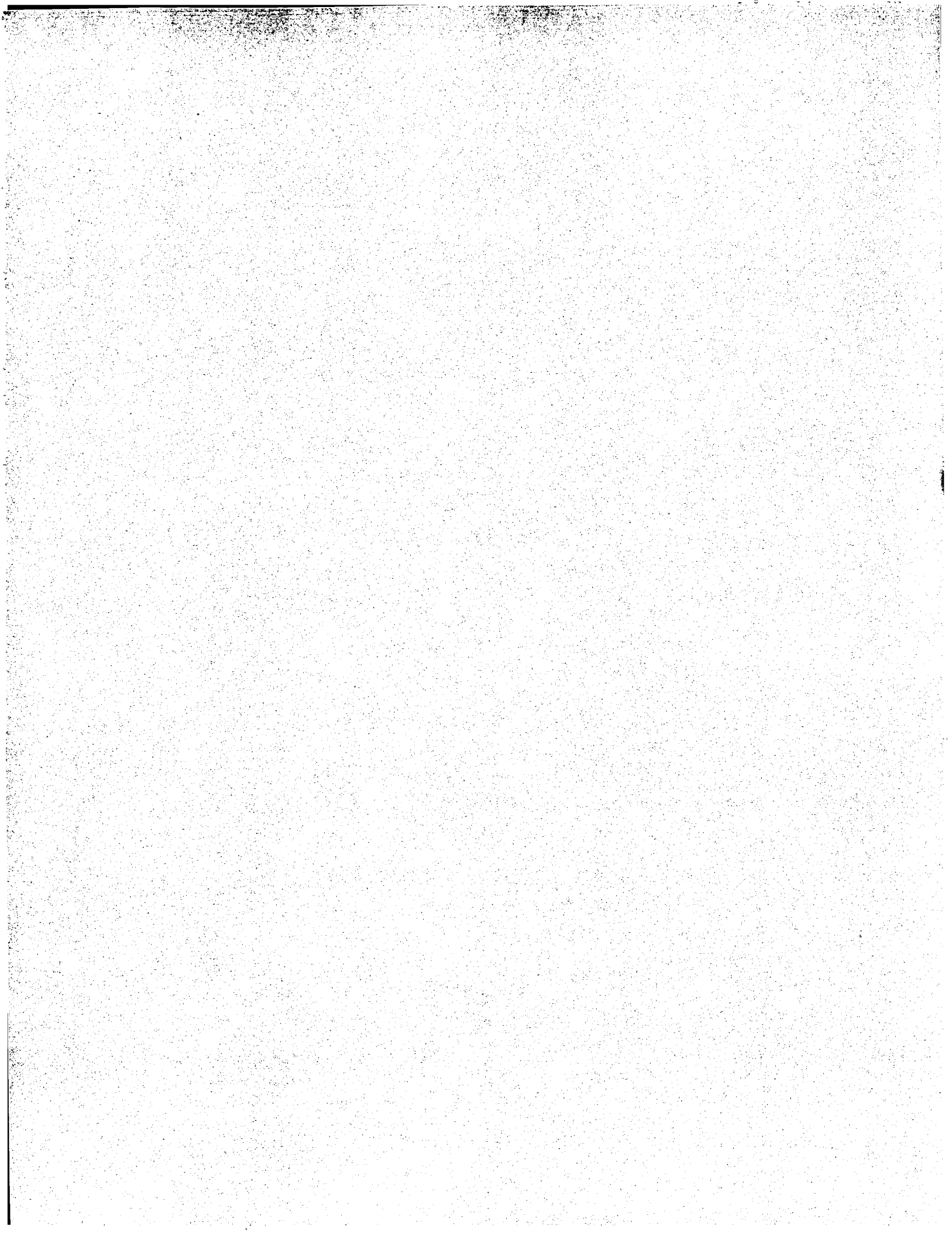
        if (try_improve(p, shuffle[m].row, win_start)) {
            improved = did_something = 1;
            update_gap_loc(gap_loc, p, shuffle[m].row, win_start, first);
        }

        while (improved && num_times++ < 20);

        /* Return memory */
        if (shuffle)
            free(shuffle);
        for(i=0; i<p->width; i++)
            if (n_gaps[i])
                free(gap_loc[i]);
        free(gap_loc);
        free(n_gaps);

        return did_something;
    }
}

```



```

/*
 * analyze_graph -- analyze the splice_graph for creation of transcripts, consen-
 * sus,
 * detection of chimeras and so forth.
 */

/*
 * $Log: analyze_graph.c,v $
 * Revision 1.11 1998/06/18 14:42:47 ariels
 * Strip "#ifdef CPROF_ALIGN" and "#ifdef IMPROVE_CPROF" lines.
 *
 * Revision 1.10 1998/04/24 15:32:22 eyal
 * Delete the call for record_err when a cycle is found in build_transcripts
 * (there is a call for that in io.c)
 *
 * Revision 1.9 1998/04/13 07:06:52 eyal
 * Remove the line 'extern char transc_db[]' (no one uses transc_db)
 *
 * Revision 1.8 1998/04/12 12:48:48 eyal
 * Replace map and cond_map with cons_map in build_consensus
 *
 * Revision 1.7 1998/04/12 11:09:30 avner
 * Print warning of too many transcripts to stdout (and not stderr).
 * Change type of map from char toint (and change name to cons_map).
 *
 * Revision 1.6 1998/02/22 21:45:25 avner
 * Don't print order of nodes in consensus when graph is of one node.
 *
 * Revision 1.5 1998/02/22 17:18:13 ariels
 * Use new error and warning reporting facility.
 * Eliminate problem_msg[].
 *
 * Revision 1.4 1998/02/21 22:37:42 avner
 * *** empty log message ***
 *
 * Revision 1.3 1998/02/17 13:13:10 eyal
 * Changing use of 30000 to MAX_TRANSC_LEN in build_transcripts.
 *
 * Revision 1.2 1998/02/11 13:13:35 avner
 * Added ChangeLog comment and static rcsid string.
 */

static char rcsid[] = "$Id: analyze_graph.c,v 1.11 1998/06/18 14:42:47 ariels Ex
p $";

#include "string.h"
#include "cprof.h"
#include "error.h"
#include "analyze_graph.h"

static int enumerate_paths_from_node(char *transcripts[MAX_TRANSC],
                                     splice_graph *, int,
                                     char *, char *, int *,
                                     int **, int *, int);

/*
 * build transcripts from the graph. this one does not accept cycles in the

```

analyze_graph.c

```

 * graph (and will return an appropriate code), and produces a table of the
 * transcripts' _themselves_ (sequences). Do we want it?
 */
int build_transcripts (splice_graph *graph,
                      char *transcripts[MAX_TRANSC],
                      int *trnsc_no, int **trnsc_map)
{
    int i, j, node_idx, temp_map[MAX_NODES];
    char print_transcript_string[MAX_TRANSC_LEN];
    char transcribe_string[MAX_TRANSC_LEN];

    *trnsc_no = 0;
    for (i = 0; i < MAX_TRANSC; i++) {
        transcripts[i] = NULL;
        transc_map[i] = (int *) malloc(sizeof(int)*MAX_NODES);
        for (j = 0; j < MAX_NODES; transc_map[i][j++] = -1);
    }
    for (j = 0; j < MAX_NODES; temp_map[j++] = -1);

    print_transcript_string[0] = transcript_string[0] = 0;
    for (node_idx = 0; node_idx < graph->size; node_idx++) {
        if (splice_graph_in_deg(graph, node_idx) == 0)
            if (enumerate_paths_from_node (transcripts,
                                           graph, node_idx, print_transcript_string,
                                           transcript_string, trnsc_no,
                                           transc_map, temp_map, 0) == -1) {
                return 0;
            }
    }
    if (*trnsc_no > MAX_TRANSC) *trnsc_no = MAX_TRANSC;
    return 1;
}

/*
 * auxiliary function for 'build_transcripts'
 */
static int enumerate_paths_from_node(char *transcripts[MAX_TRANSC],
                                     splice_graph *graph, int node_idx,
                                     char *print_string,
                                     char *transcript_string,
                                     int *trnsc_no, int **trnsc_map,
                                     int *temp_map, int map_index)
{
    int prevlen = strlen(print_string);
    int prevlen2 = strlen(transcript_string);
    splice_node *node = graph->node_list[node_idx];
    int nbr, i;

    if (node->marked) return -1;
    node->marked = 1;

    sprintf(print_string, "%s\n%d%s", print_string, node_idx, node->seq);
    sprintf(transcript_string, "%s%s", transcript_string, node->seq);
    temp_map[map_index] = node_idx;

    if (node->out_degree == 0) {
        if (*trnsc_no < MAX_TRANSC)
            {

```

analyze_graph.c

```

if ((transcripts["trnsc_no"] =
    (char *)strdup(transcript_string)) == NULL)
    err("enumerate_paths_from_node: memory failure");
    "trying to allocate for 'transcripts[%d]'", "trnsc_no");
for(i=0; i<map_index; i++)
    transc_map["trnsc_no"][i] = temp_map[i];
transc_map["trnsc_no"][map_index+1] = -1;
}
else
    printf ("Not adding transcript %d to alignment list\n",
        "trnsc_no");
    (*trnsc_no)++;
}
else
    for (nbr = 0; nbr < node->out_degree; nbr++)
        if ((node->out_neighbors[nbr].is_ximeric)
            if (enumerate_paths_from_node (transcripts,
                graph, node->out_neighbors[nbr].idx,
                print_string, transcript_string,
                trnsc_no, transc_map, temp_map,
                map_index+1)
                == -1)
                return -1;
        print_string(prevlen) = 0;
        transcript_string[prevlen2] = 0;
        temp_map[map_index] = -1;
        node->marked = 0;
        return 0;
    }

/*=====
/*
* Find a topological order of the graph, and returns 1/0 for unique/nonunique
* such order, or -1 for no order at all (i.e. when the graph is cyclic)
*/
int build_consensus (splice_graph *graph, char *consensus, int cons_map[],
    int *nodes_order_arr)
{
    splice_node *node;
    int marked[MAX_NODES], i;
    int candidate_node, good_candidate, num_good_candidates, ordinal, unique_order,
        base_in_map=0, base_in_node, nbr;

    bzero ((char*)marked, sizeof (marked));
    for(i=0; i<MAX_NODES; nodes_order_arr[i++] = -1);

    for (ordinal = graph->size-1, unique_order=1; ordinal >= 0; ordinal--)
    {
        for (candidate_node=0, num_good_candidates=0;
            candidate_node < graph->size; candidate_node++)
        {
            if (marked[candidate_node]) continue;
            node = graph->node_list[candidate_node];
            for (nbr = 0;
                nbr < node->out_degree && marked[node->out_neighbors[nbr].idx];
                nbr++)
            {
                if (nbr==node->out_degree) /* candidate_node is a good candidate */
                {
                    num_good_candidates++;

```

```

        }
        good_candidate = candidate_node;
    }
    if (num_good_candidates==0)
        return -1;
    if (num_good_candidates > 1)
        unique_order=0;
    nodes_order_arr[ordinal] = good_candidate;
    marked[good_candidate] = 1;
}

if (graph->size > 1)
    printf ("order of nodes in the consensus is: ");
for (ordinal = 0, consensus[0]='0'; ordinal < graph->size; ordinal++)
{
    strcat (consensus, graph->node_list[nodes_order_arr[ordinal]]->seq);
    if (graph->size > 1)
        printf ("%d ", nodes_order_arr[ordinal]);
    for (base_in_node=0;
        base_in_node < strlen
            (graph->node_list[nodes_order_arr[ordinal]]->seq);
        base_in_node++)
        cons_map[base_in_map++] = ordinal;
    }
    printf("\n");
    cons_map[base_in_map] = -1;
    return unique_order;
}

```

```

/* main function for creating the splice graph */
/*
 * $Log: build_graph.c,v $
 * Revision 1.57 1998/06/24 07:20:04 eyal
 * Add a call to new_adl_to_czm incase node_path_mode=y.
 *
 * Revision 1.56 1998/06/16 20:58:45 avner
 * 1. adding global 'phase_no' which is 1 when building the variants
 * and 2 when building the splice graph.
 * 2. getting rid of #ifdef CPROF_ALIGN sections.
 *
 * Revision 1.55 1998/06/15 08:04:44 eyal
 * fix bug in remove_dirty_nodes
 *
 * Revision 1.54 1998/06/14 05:50:59 eyal
 * 1. Add calls for functions cut_est_head cut_est_tail in remove_dirty_tails
 *
 * Revision 1.53 1998/05/13 13:20:29 eyal
 * Remove #ifdef IDENTITY_PHASE2 (it is steel
 * in active when phase2_bound is zero)
 *
 * Revision 1.52 1998/04/28 09:57:40 ariels
 * Switch to use bsort(), an (idiotic) bubble-sort. This because qsort()
 * produces different results on different platforms, since many nodes
 * can have the same thicknesses!
 *
 * Revision 1.51 1998/04/27 14:37:11 eyal
 * Change contig_name and cluster_name to be char [] instead of char *
 *
 * Revision 1.50 1998/04/25 17:02:36 avner
 * handle rp_mode in a way that allow 'secondary' mode to work ok. add
 * global variable 'need_rmnode' which is 0 if graph is valid after regular
 * mode, 1 if big-intersection, and 2 if cyclic.
 *
 * Revision 1.49 1998/04/24 15:36:39 eyal
 * Make phase one of rp_mode=s run in mode overlap, and phase two in local
 *
 * Revision 1.48 1998/04/23 10:43:08 eyal
 * Add a call for cut_dirty_overhangs.
 *
 * Revision 1.47 1998/04/20 14:51:29 eyal
 * 1. Call cprof_init_dirty when needed
 * 2. use cprof_is_dirty instead of cprof_is_dirty_tmp
 *
 * Revision 1.46 1998/04/20 07:37:18 avner
 * make remove_dirty_nodes work iterationwise for simplification of the dirty
 * influence. Doesn't seem that this the solution does the trick.
 *
 * Revision 1.45 1998/04/19 08:05:17 eyal
 * Add a call to find_nodes_types at the end of build_graph.
 *
 * Revision 1.44 1998/04/18 21:46:37 avner
 * bypass use of cprof_init_dirty due to a bug.
 *
 * Revision 1.43 1998/04/15 15:07:04 ariels
 * Call cprof_init() with appropriate arguments for ests or RNAs.
 *
 * Revision 1.42 1998/04/15 11:46:37 avner
 * make the call to 'cprof_init_dirty' only when dirty_tails flag is set to 'a'.

```

build_graph.c

```

* Revision 1.41 1998/04/14 12:41:37 ariels
* Incorporate calls to cprof_init_dirty().
*
* Revision 1.40 1998/04/14 10:11:20 eyal
* make rp_mode=s run in local overlap mode
*
* Revision 1.39 1998/04/14 07:33:15 avner
* add calls to cprof_init_dirty, and cprof_init_clean to init a sequence which
* has suspected to be dirty segments in it, or is expected to be cleaner than
* normal (RNA). Right now the function are written here and do cprof_init; just
* allow compilation.
* add clear_err() in the rebuild case.
*
* Revision 1.38 1998/04/08 05:43:05 eyal
* 1. Adding function rebuild_splice_graph_in_rp_mode and a call for it
* 2. Fix memory leak - freeing the graph when adl_to_czm fails
*
* Revision 1.37 1998/04/05 12:18:45 eyal
* Adding function remove_dirty_nodes and a call for it
*
* Revision 1.36 1998/03/29 20:45:35 avner
* 'est_table_seq' is called upon whenever the sequence with which we _assemble
* (as opposed to other post-process stuff) need to be used.
*
* Revision 1.35 1998/03/13 14:25:32 avner
* changing calls to functions that changed their names.
*
* Revision 1.34 1998/02/16 08:42:33 eyal
* Merging with version 1.31.1 (with the call to the new cprof_init).
*
* Revision 1.33 1998/02/15 22:25:01 avner
* add global 'old_graph_size' for detection of edges that are formed not
* through a broken-overlap (between sequences/variants).
*
* Revision 1.32 1998/02/14 17:29:29 avner
* Change call to 'adl_to_czm' when CPROF_ALIGN (add 'entering_profile').
*
* Revision 1.31.1.1 1998/02/12 15:17:45 ariels
* Call cprof_init() with second 'identifier' argument, thus requires
* new version of cprof.[ch].
*
* Revision 1.31 1998/02/11 11:27:46 avner
* 'separating in peace' operation. The old build_graph was separated to three
* different sources (build_graph, analyze_graph and repeats). What is left
* in this source is what indeed involves the (high-level) creation of the
* splice_graph.
*
* Revision 1.30 1998/02/09 06:46:30 avner
* cancel the last change temporarily to allow compilation. Call 'order_ests'
* without expecting a failure.
*
* Revision 1.29 1998/02/08 10:44:54 avner
* call 'cprof_init' with an added parameter (the index of the est by which the
* profile
* is created).
*
* Revision 1.28 1998/02/05 09:54:17 ariels
* Change splice_graph var to 'graph' in build_graph (splice_graph is also

```

build_graph.c


```
* a type, which isn't a nice thing to do!).
*
* Revision 1.27 1998/02/04 15:30:09 eval
* Fix FUN bug - free(sequence) in rp_mode.update_splice_graph(...) was under
* #ifdef CPROF_ALIGN instead of #ifdef CPROF_ALIGN.
*
* Revision 1.26 1998/02/01 07:06:59 eval
* Add a call for align_data_list_identity_filter in the first phase,
* (and under #ifdef in phase two).
*
* Revision 1.25 1998/01/25 11:07:06 eval
* Fixing bug in rp_mode.update_splice_graph in CPROF_ALIGN mode.
* sequence was created from profile instead of nrs->profile.
*
* Revision 1.24 1998/01/21 07:52:04 ariels
* Put err() in scope of prototype (add #include "error.h")
*
* Revision 1.23 1998/01/14 10:05:06 eval
* Add support for rna_test_mode.
*
* Revision 1.22 1998/01/12 11:50:28 eval
* Fixing warnings of compilations
*
* Revision 1.21 1998/01/11 13:53:22 ariels
* Fix error reporting to use err()
*
* Revision 1.20 1998/01/08 17:30:00 avner
* free 'variant_seq' in 'check_repeats_within_variants'.
*
* Revision 1.19 1998/01/08 13:53:02 ariels
* Correctly merge second memory leak fix (rev. 1.7.1.2) into main trunk.
*
* Revision 1.18 1998/01/07 23:57:49 avner
* further modifications to use cprof_align in the right manner.
* Instead of creating 'variant_seq' in build_splice_graph_from_variant_graph,
* just take it from the node->seq.
* change in print format of get_nrs_list: it's locations are now absolute.
* some more documentation and code-cleaning.
*
* Revision 1.17 1998/01/07 17:41:39 eval
* Fixing CPROF merging bugs
*
* Revision 1.16 1998/01/07 12:12:17 ariels
* Merged memory-leak fix (see revision 1.7.1). Fixed some problems when
* compiling with -DCPROF_ALIGN.
*
* ****THIS REVISION DOES NOT COMPILE!!!! There are severe problems with
* all the code enabled by -DCPROF_ALIGN.
*
* Revision 1.15 1998/01/07 11:01:44 eval
* adding RP-MODE functions
*
* Revision 1.14 1998/01/02 13:16:36 avner
* make sure short nodes are not created: when a sequence in 'process_nrs'
* is smaller than <indep_node_len>, we simply neglect it (these is the
* analogue of what we do with no rp_mode. Is it the best we can do? perhaps...).
* Besides, add identity report for repeats detected in 'get_nrs_list'.
* This should help us see if there is a nice separation between 'homologues
* repeats' and identical ones.
*

```

build_graph.c

```
* Revision 1.13 1997/12/31 15:05:22 eval
* rp_mode
*
* Revision 1.12 1997/12/29 16:02:28 eval
* RP_MODE work
*
* Revision 1.11 1997/12/22 07:39:23 eval
* Calling check_repeats for variant0 when check_repeats!='n'.
* and more rp_mode changes.
*
* Revision 1.10 1997/12/21 22:40:11 avner
* check_repeats returns 1 if straight_repeat found. 0 if no repeat,
* and -1 otherwise (inversed but not straight).
*
* Revision 1.9 1997/12/21 13:38:26 eval
* Start working on rp_mode (not compiled yet)
*
* Revision 1.8 1997/12/20 21:23:17 avner
* continue rp_mode work.
*
* Revision 1.7 1997/12/17 06:14:01 eval
* Adding a check for repeats in variant 0.
*
* Revision 1.6 1997/12/15 21:40:25 avner
* addition of documentation
*
* Revision 1.5 1997/12/14 22:36:04 avner
* changing the calls to 'apply_transformation_?' function (no more penalty
* accounting)
*
* Revision 1.4 1997/12/10 22:08:28 avner
* *** empty log message ***
*
* Revision 1.3 1997/12/10 22:07:10 avner
* adding the feature of multihit check among the variants (via Hilltops)
*
* Revision 1.2 1997/11/30 07:41:45 ariels
* Added ChangeLog comment and static rcsid.
*
* */
static char rcsid[] = "$Id: build_graph.c,v 1.57 1998/06/24 07:20:04 eval Exp $";

#include <assert.h>
#include "build_graph.h"
#include "io.h"
#include "string.h"
#include "cprof.h"
#include "error.h"
#include "repeats.h"

int old_graph_size, need_rp_mode, phase_no;

extern int indep_node_len, local_olap_mode, phase_id_bound, phase2_id_bound;
extern char check_repeats, rp_mode, p_graph_operations, dirty_tails;
extern char node_path_mode;
extern int gb_ver, directed_cycle;
extern char config_name[], cluster_name[];

```

build_graph.c


```

/* prototypes of static functions */
static splice_graph *build_variant_graph(void);
static splice_graph *variant_graph_to_splice_graph(splice_graph *);
static void sort_variant_graph(splice_graph *variant_graph);
static int variant_node_compar(const void** node1, const void** node2);
static void prepare_seqs(splice_graph *graph);
static splice_graph *rebuild_splice_graph_in_rp_mode(splice_graph *graph,
    splice_graph *variant_graph);
static int remove_dirty_nodes(splice_graph *graph);

```

```

/* ===== */
/*
 * build_graph ::
 * The main assembly procedure. partitions the sequences such that each
 * set induces one assembly sequence (which we call 'varinat').
 * Then go over all variants and create progressively a graph that
 * serves as the model for the assembly of the variants seen so far.
 * The output of this function is the final splice_graph of the current
 * contig, or NULL in case of a failure.
 */

```

```

splice_graph *build_graph(void)
{
    int original_local_mode;
    splice_graph *variant_graph,*graph;
    printf(" ----- assembly with alternative-splicing begins-----\n\n");
    need_rpmode = 0;

```

```

/* first phase */
phase_no = 1;
if ((variant_graph = build_variant_graph()) == NULL)
    return NULL;
if (variant_graph->size == 1) /* no need to work further */
    return variant_graph;
original_local_mode = local_olap_mode;
if (rp_mode == 'n' || rp_mode == 't' || rp_mode == 's')
    /* we want olap mode in this phase if it is not rp_mode */
    local_olap_mode=0;

```

```

sort_variant_graph(variant_graph);
graph__print(variant_graph);
prepare_seqs(variant_graph);
if (check_repeats=='v' || check_repeats=='m')
    check_repeats_within_variants(variant_graph);
if (check_repeats=='m')
    check_multiple_hits_in_variants(variant_graph);

```

```

/* second phase */
phase_no = 2;
if (rp_mode == 'y' || rp_mode == 'b')
    graph = rpmode_variant_graph_to_splice_graph(variant_graph);
else
    graph = variant_graph_to_splice_graph(variant_graph);

```

```

if (rp_mode == 's') { /* s for secondary */
    if (need_rpmode = splice_graph_need_rebuild(graph)) {
        /* rp_mode='y'; we should mark for the transcript phase that the graph
           is built in rp_mode */

```

build_graph.c

```

    local_olap_mode = original_local_mode; /* we want to run in local_mode */
    graph = rebuild_splice_graph_in_rp_mode(graph,variant_graph);
}
splice_graph_free(variant_graph);
if (graph != NULL) {
    find_nodes_types(graph);
    find_cycle(graph,1); /* 1 for printing the cycles */
}
local_olap_mode=original_local_mode;
return graph;
}

```

```

int splice_graph_need_rebuild(splice_graph *graph)
{
    int rc=0;
    do {
        if(graph == NULL) {
            rc =1;
            break;
        }
        if(find_cycle(graph,0)) {
            rc =2;
            break;
        }
    } while(0);
    return rc;
}

```

```

splice_graph *rebuild_splice_graph_in_rp_mode(splice_graph *graph,
    splice_graph *variant_graph)
{
    splice_graph_free(graph);
    clear_err();
    printf("rp-mode work done with %s.%s \n",
        cluster_name,contig_name);
    printf("because of %s\n", (need_rpmode==1)?
        "big-intersection":"cyclic graph");
    return rpmode_variant_graph_to_splice_graph(variant_graph);
}

```

```

/* ===== */
/* build_variant_graph:
 * partition the sequences into sets which are assembled 'with no hassles',
 * that is, to one continuous sequences (one node). General scheme: scan
 * the sequences progressively, and the current sequence joins a
 * variant which tolerate him, i.e. remains simple-assembled with it.
 * if there is no such variant, this sequence is the only element of a
 * new variant.
 * It is noteworthy that we don't really create a graph, but rather a list
 * (of variants). A more suitable structure would be a list of profiles,
 * but it is easier to use support we have from the (more complicated)
 * second, and in order to use 'update_splice_graph', the graph structure
 * remains.
 */

```

build_graph.c


```

printf(">variant %d      #LN %d\n",
       variant.splice_node___len(variant_node));
for (i= 0, graph_align_list=NULL; i<graph->size; i++) {
    /* for every node, see how it aligns with the next variant */
    node = graph->node_list[i];
    align_data_list = align_node_to_cprof(node,variant_node->profile);
    merge_ok = merge_node_information (node_align_list, &graph_align_list);
    align_data_list_free(node_align_list);
    if (!merge_ok)
        {
            splice_graph_free(graph);
            align_data_list_free(graph_align_list);
            return NULL;
        }
    align_data_list___print(graph_align_list);
    if (node_path_mode == 'y')
        czm = new_adl_to_czm(graph_align_list,splice_node___len(variant_node),
                           variant_node->profile,graph);
    else
        czm = adl_to_czm(graph_align_list,splice_node___len(variant_node),
                           variant_node->profile);
    if (czm==NULL) {
        splice_graph_free(graph);
        return NULL;
    }
    if (p_graph_operations == 'y' || p_graph_operations == 'v')
        czm___print(czm);
    /* update */
    update_splice_graph_ok = update_splice_graph(graph,variant_node->profile,
                                                  czm);
    align_data_list_free(czm);
    if (!update_splice_graph_ok)
        {
            splice_graph_free(graph);
            return NULL;
        }
    else
        {
            if (p_graph_operations == 'y' || p_graph_operations == 'v')
                printf("splice-graph after taking the %d'th variant\n",variant);
            graph___print(graph);
            printf("=====\n");
        }
    if (dirty_tails == 'y') { /* too bad we have to do it iterationwise */
        if (remove_dirty_nodes(graph)) {
            printf("graph after removing dirty nodes\n");
            graph___print(graph);
        }
    }
    return graph;
}

```

```

/*=====
/*
    * update_splice_graph
    */

int update_splice_graph(splice_graph *graph, cprof entering_profile,
                        align_data *czm)
{
    splice_node *node;
    align_data *west_bank, *east_bank;

    int i, more_than_one_segment = 0;

    /*
    we go over the custody zones map list, and at every cut point
    we apply the appropriate transformation by the following scheme:

    ---1---|----FREE_SEG-----:
    (a) the segment of 1 is the node's suffix
        (i) 1 has out degree 0      append to the node the next segment
        (ii)1 has out degree>0     create a node for -1 segment and
                                   an edge from 1 to -1
    (b) the segment of 1 is not it's suffix
        split 1 to the segment here and the suffix from here on
        an edge between this splice's parts.
        create a node for next segment and an edge from
        the left part of 1 to it

    ----FREE_SEG-----|----1-----:
    (a) the segment of 1 is the node's prefix
        (i) 1 has in degree 0      add FREE_SEG segment to node 1
        (ii)1 has in degree>0     create a node for FREE_SEG
                                   segment and an edge from
                                   FREE_SEG to 1
    (b) the segment of 1 is not it's prefix
        split 1 to the segment here and the prefix upto here
        an edge between this splice's parts.
        create a node for -1 segment and an edge from -1
        to the right part of 1

    ---1---|----2---
    (a) 1 is node's suffix, 2 is the node's prefix.
    an edge from 1 to 2 should exist. if it already is, either do nothing
    or strengthen the likelihood of this edge
    (b) 1 is not the node's suffix, 2 is.
    split 1 accordingly. not.
    */

    while (get_next_zone_boundary(&czm,&west_bank,&east_bank)) {
        more_than_one_segment = 1;
        if (west_bank->node_id >= 0 && east_bank->node_id == FREE_SEG) {
            if (!apply_transformation_A(graph, czm, west_bank, east_bank, entering_pro

```

```

file))
    return 0;
}
else
    if (west_bank->node_id == FREE_SEG && east_bank->node_id >= 0) {
        if (!apply_transformation_B(graph, czm, west_bank, east_bank, entering_pro
file))
            return 0;
    }
    else
        if (west_bank->node_id >= 0 && east_bank->node_id >= 0) {
            if (!apply_transformation_C(graph, czm, west_bank, east_bank, entering
_profile))
                return 0;
        }
    else
        err("update_splice_graph : internal error, "
            "call amit/avmer (unknown case)");
}

if (!apply_transformation_single
    (graph, west_bank, entering_profile, more_than_one_segment))
    return 0;

/* now update the nodes with the right data */
for(i=0; i<graph->size; i++) {
    node = graph->node_list[i];
    if (node->need_to_update==1)
        update_node(node, entering_profile);
    node->need_to_update = 0;
}

unify_needed_nodes(graph);
return 1;
}

/*****
* The compar function for the qsort.
*****/
int variant_node_compar(const void** node1, const void** node2)
{
    int depth1, depth2;

    depth1 = ((splice_node**)node1)->variant_depth;
    depth2 = ((splice_node**)node2)->variant_depth;

    if (depth1 == depth2) return 0;
    if (depth1 > depth2) return -1;
    if (depth1 < depth2) return 1;
}

/**** NOTREACHED */

/*****
* Sort the node of the variant graph according to the node depth.
*****/
void sort_variant_graph(splice_graph *variant_graph)
{

```

build_graph.c

```

    bsort(variant_graph->node_list,
        variant_graph->size,
        sizeof(splice_node*),
        &variant_node_compar);
}

/*****
* * prepare_seqs : for every node, produce the string representing
* * it's profile.
* */
static void prepare_seqs(splice_graph *graph)
{
    int i, dummy;

    for (i=0; i<graph->size; i++)
        graph->node_list[i]->seq =
            cprof_compute_assembly(graph->node_list[i]->profile, 0, &dummy, &dummy);
}

static int remove_dirty_nodes(splice_graph *graph)
{
    int i, est_idx, nbr, start, end, cut_len, deleted=0;
    splice_node *node;

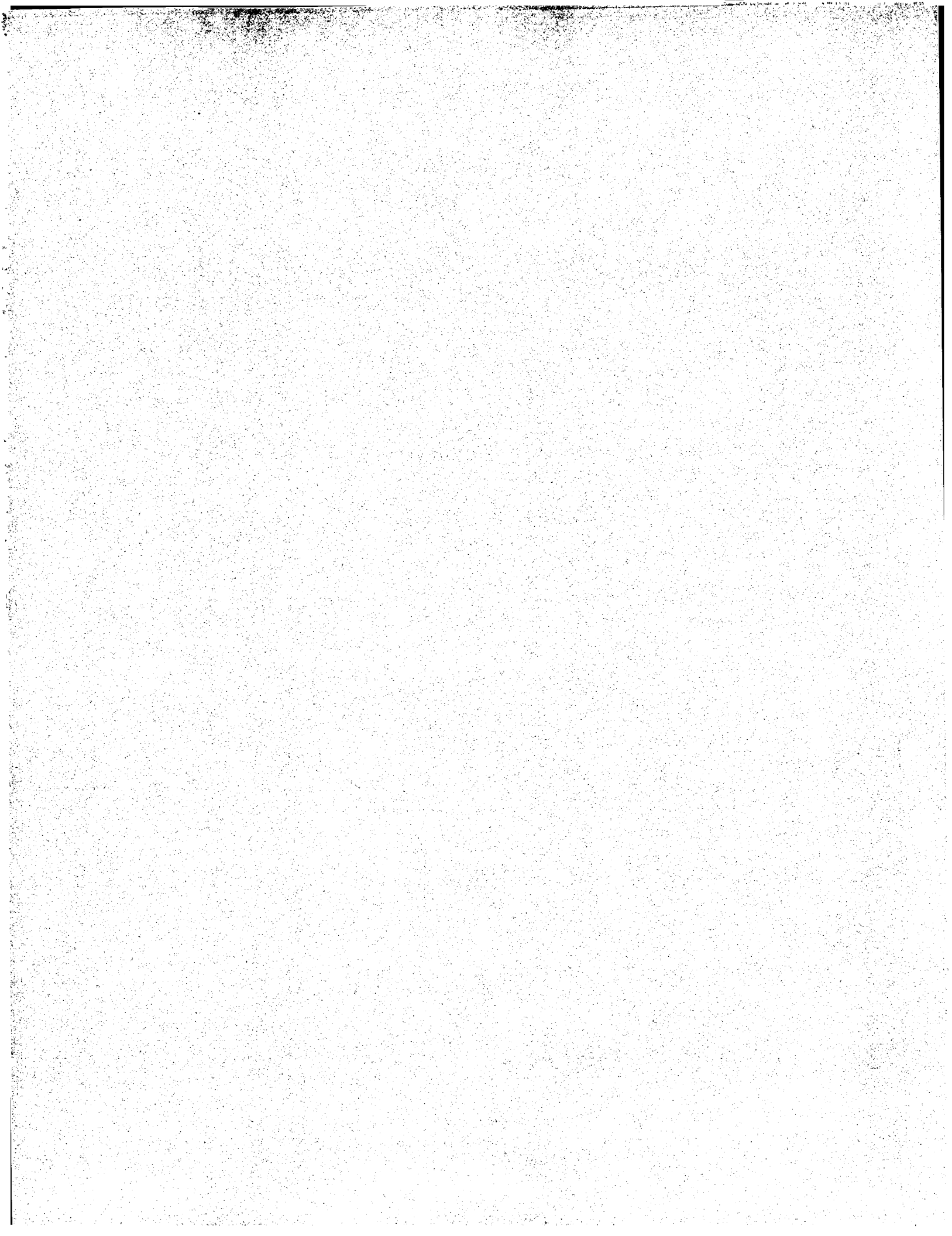
    for(i=graph->size-1; i>=0; i--) {
        node = graph->node_list[i];
        if (cprof_is_dirty(node->profile)) {
            if (node->out_degree!=0 && node->in_degree!=0) {
                printf("Node %d is dirty but not a leaf!\n", i);
            }
        }
        else {
            cprof_get_est_start_end(node->profile, -1, node->profile->id(0), &start, &en
d);
            assert(cprof_ids(node->profile, &est_idx, 1) == 1);
            printf("deleting node %d because it is supported only by est %d's %s ",
                i, est_idx,
                end < est_table__first_clean(est_idx)? "head": "tail");
            printf("(cleaning %d %d, supporting segment %d %d)\n",
                est_table__first_clean(est_idx),
                est_table__first_dirty(est_idx), start, end);
            cut_len = node->profile->len;
            for(nbr=0; nbr<node->out_degree; nbr++) {
                graph->node_list[node->out_neighbors[nbr].idx]->in_degree--;
            }
            graph__delete_node(graph, i, 1); /*1 because we want to free the profile*/
        }
    }

    /* cutting the est sequence in the est table */
    if (end < est_table__first_clean(est_idx))
        cut_est_head(est_idx, cut_len, graph);
    else
        cut_est_tail(est_idx, cut_len, graph);
    deleted=1;
}

```

build_graph.c

```
    )  
    if (deleted)  
        unify_needed_nodes (graph);  
    return deleted;  
}
```



```

/*
 * cprof.c -- Support and scoring routines for counted profiles
 *
 * Apart from access routines for counted profiles, counted profiles
 * can also tell you how many gap opens should be paid for when a gap
 * is opened at a given link (this makes sense).
 *
 * ariels@compugen 9/10/97
 */

/*
 * $Log: cprof.c,v $
 * Revision 1.29 1998/06/18 14:44:09 ariels
 * Strip "#ifdef CPROF_ALIGN" and "#ifdef IMPROVE_CPROF" lines.
 *
 * Revision 1.28 1998/06/15 12:12:58 eval
 * Add function cprof_get_cprof_start_end
 *
 * Revision 1.27 1998/06/14 05:48:23 eval
 * Add the function cprof_pos_ids.
 *
 * Revision 1.26 1998/05/04 08:01:55 ariels
 * Add cprof_ids() (which see)
 *
 * Revision 1.25 1998/05/03 08:46:53 ariels
 * Fix cprof_update() bug regarding incorrect updating of the 'pos'
 * field. No longer use cprof_get_id_pos(), since its semantics are
 * simply "wrong" for getting locations on a particular pass in the
 * multiple alignment.
 *
 * Revision 1.24 1998/04/23 14:37:06 eval
 * Fix bug in cprof_get_est_start_end - and was one ahead
 *
 * Revision 1.23 1998/04/21 09:17:34 eval
 * Add functions cprof_first_clean and cprof_last_clean
 *
 * Revision 1.22 1998/04/15 15:05:22 ariels
 * Add very_clean fields to cprof for dealing with "very clean" sequences
 * in the multiple alignment.
 *
 * Revision 1.21 1998/04/15 09:23:03 ariels
 * Correctly call the new cprof_node_char().
 *
 * Revision 1.20 1998/04/14 14:59:15 ariels
 * Fix ANOTHER bug (thinks) in blank squishing in cprof_update().
 *
 * Revision 1.19 1998/04/14 13:54:45 ariels
 * Fix bug (found by Eyal) with blank squishing in cprof_update()
 *
 * Revision 1.18 1998/04/14 12:40:58 ariels
 * Really add cprof_init_dirty()!
 *
 * Revision 1.17 1998/04/14 09:42:53 ariels
 * New format for multiple alignment to allow "dirty" positions.
 * New routine cprof_init_dirty() allows sequences with dirty tails.
 * Fix cprof_update() problem of completely empty sequences in the
 * multiple alignment (having gaps DOESN'T counter as empty!).
 *
 * Revision 1.16 1998/04/12 07:00:28 eval
 * 1. Changing the begin parameter of the call for cprof_get_id_pos from pl->pos

```

cp Prof.c

```

[w1+j] to
 * pl->pos[w1+j]-1.
 * 2. Add to cprof_verify a verification that there are no empty rows in the cpr
 * of
 *
 * Revision 1.15 1998/04/08 15:24:37 ariels
 * Unify with 1.9.1 branch for better (but still hacked) handling of
 * 'N's.
 *
 * Fix cprof_update() failure and incorporate Eyal's fix for empty
 * sequences.
 *
 * Restore cprof_get_id_pos() to its original (correct) semantics.
 *
 * Revision 1.14 1998/04/08 08:04:05 eval
 * Fix bug (?) in cprof_get_id_pos
 *
 * Revision 1.13 1998/04/05 06:57:02 eval
 * 1. Adding functions cprof_is_dirty and cprof_get_est_start_end
 * 2. Fixing bug in cprof_get_seq_location - don't return an entry number
 * corresponding to a space.
 *
 * Revision 1.12 1998/03/24 09:04:04 eval
 * Add a call to eliminate_squished_seqs(...) in cprof_update,
 * Ariel check to see if it is OK.
 *
 * Revision 1.11 1998/03/01 10:20:04 ariels
 * Change cprof_get_id_pos() to consider a sequence in multiple alignment
 * as terminated at first space.
 *
 * Revision 1.10 1998/02/26 11:17:00 ariels
 * Fix cprof_update() bug for sequence-start field.
 *
 * Revision 1.9.1.1 1998/02/23 14:28:08 ariels
 * Klugey 'N'-counting code.
 * Count everything in the cprof twice, once with N's transformed to
 * random letters (for alignment) and once without the N's. Keep
 * lowercase letters for the randomised N's, so we can still improve
 * cprofs, yet get "correct" output.
 *
 * Revision 1.9 1998/02/22 07:14:44 ariels
 * Add cprof_get_seq_first_location function to return position of
 * beginning of a sequence's pass from a cprof.
 *
 * Revision 1.8 1998/02/16 08:11:05 ariels
 * Keep track of sequence identifiers in the multiple alignment.
 *
 * Revision 1.7 1998/02/05 09:55:41 ariels
 * Fix small memory leak in cprof_update().
 *
 * Revision 1.6 1998/02/01 12:15:06 ariels
 * Keep track of multiple alignment in each profile.
 *
 * Almost works, but not quite -- there are still difficulties stemming
 * from different semantics of the multiple alignment and the "link"
 * field.
 *
 * Revision 1.5 1998/01/08 13:36:41 ariels
 * Make parameter block (prm) static in align_cprof.c, rather than an
 * externally supplied set of values.

```

cp Prof.c

```

* Move cprof_print() and cprof_compute_assembly() to align_cprof.c,
* replacing them by a maximal-likelihood version.
*
* Revision 1.4 1997/12/31 10:36:38 ariels
* Removed 'cost' field from profiles; instead of it are a pair of static
* variables in align_cprof.c which hold the location scores for the pair
* of profiles being aligned.
*
* Removed the dead function cprof_shorten().
*
* Revision 1.3 1997/12/17 10:24:01 ariels
* Added 'score' field to profiles (used to speed up alignments)
*
* Revision 1.2 1997/11/30 07:46:29 ariels
* Added ChangeLog comment and static rcsid.
*/

```

```
static char rcsid[]="$Id: cprof.c,v 1.29 1998/06/18 14:44:09 ariels Exp $";
```

```

#include "cprof.h"
#include "cprof_align.h"

#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <stdio.h>
#include <values.h>
#include "error.h"

#include <assert.h>

#ifdef NDEBUG
#define CHECK(x)
#else
#define CHECK(p) cprof_verify(p)
#endif

char *bases = "-ACGT";

enum base_names {e_gap, e_a, e_c, e_g, e_t};

/* Debugging routine, thus no prototype in cprof.h... */
void cprof_dump(cprof);

unsigned char *cprof_seq_ptr(cprof p, int i, int seq)
{
    assert(0 <= i && i < p->len);
    assert(0 <= seq && seq < p->width);
    return p->seq + seq*p->len + i;
}

/* Given a LOCATION in est ID assign START and END with the start and
* end locations of the pass which begins after LOC in the est.
* return 0 if there is no pass of ID which begin after LOC, else 1.
*/

```

```

*/
int cprof_get_est_start_end(cprof p, int loc, identifier_t id, int *start,
                           int *end)
{
    int i, j, started, ended;
    unsigned char ch;
    int pass_idx=-1, best_loc = MAXINT;

    for(i=0; i<p->width; i++) {
        if (p->id[i] == id && p->pos[i] > loc && p->pos[i] < best_loc) {
            best_loc = p->pos[i];
            pass_idx = i;
        }
    }

    if (pass_idx != -1) {
        *start = p->pos[pass_idx];
        loc = p->pos[pass_idx];
        started = 0;
        for(i=0; i<p->len; i++) {
            ch = *cprof_seq_ptr(p, i, pass_idx);
            if (CPROF_IS_BLANK(ch) && started) {
                break;
            }
            if (! CPROF_IS_BLANK(ch) && CPROF_LETTER_NO(ch) > 0) {
                ++loc;
                started = 1;
            }
            *end = loc-1;
        }
        else {
            *start = *end = -1;
        }
        return pass_idx != -1;
    }

    /* Given a LOCATION in est ID assign START and END with the start and
    * end locations in the cpro aligned to the pass which begins after LOC
    * in the est.
    * return 0 if there is no pass of ID which begin after LOC, else 1.
    */
    int cprof_get_cprof_start_end(cprof p, int loc, identifier_t id, int *start,
                                   int *end)
    {
        int i, j, started, ended;
        unsigned char ch;
        int pass_idx=-1, best_loc = MAXINT;

        for(i=0; i<p->width; i++) {
            if (p->id[i] == id && p->pos[i] > loc && p->pos[i] < best_loc) {
                best_loc = p->pos[i];
                pass_idx = i;
            }
        }

        if (pass_idx != -1) {
            loc = p->pos[pass_idx];
            started = 0;
            for(i=0; i<p->len; i++) {

```



```

ch = *cprof_seq_ptr(p,i,pass_idx);
if (CPROF_IS_BLANK(ch) && started) {
    break;
}
if (! CPROF_IS_BLANK(ch)) {
    if (! started)
        *start = 1;
    if (! CPROF_IS_BLANK(ch) && CPROF_LETTER_NO(ch) > 0) {
        ++loc;
    }
    *end = i-1;
}
else {
    *start = *end = -1;
}
return pass_idx != -1;
}

int cprof_is_all_dirty(cprof p)
{
    int i, j;
    unsigned char ch;
    for(i=0; i<p->width; i++)
        for(j=0; j<p->len; j++) {
            ch = *cprof_seq_ptr(p,j,i);
            if (! CPROF_IS_BLANK(ch) && ! CPROF_IS_DIRTY(ch))
                return 0;
        }
    return 1;
}

int cprof_is_dirty(cprof p)
{
    return (p->width == 1) && cprof_is_all_dirty(p);
}

```

```

/*
 * Given a POSITION in CPROF, return the first location aligned to it in
 * the identified sequence which appears (strictly) after BEGIN. (-1 if
 * not found)
 */
int cprof_get_id_pos(cprof p, int pos, identifier_t id, int begin)
{
    int i, loc, least_loc;
    int j;
    unsigned char ch;

    least_loc = MAXINT;
    for(i=0; i<p->width; i++)
        if (p->id[i] == id) {
            for(j=0; loc=p->pos[i]; j<pos; j++) {
                ch = *cprof_seq_ptr(p,j,i);
                if (! CPROF_IS_BLANK(ch) && CPROF_LETTER_NO(ch) > 0)
                    ++loc;
            }
        }
}

```

```

}
if (p->pos[i] <= loc && loc < least_loc && begin < loc)
    least_loc = loc;
}
return (least_loc == MAXINT ? -1 : least_loc);
}

/*
 * Given a location in sequence ID, return the location in cprof p to
 * which it is aligned.
 */
int cprof_get_seq_location(cprof p, identifier_t id, int loc)
{
    int i, j, seq_loc;
    unsigned char ch;

    for(i=0; i<p->width; i++)
        if (p->id[i] == id)
            for(j=0; seq_loc=p->pos[i]; seq_loc<loc && j<p->len; j++) {
                ch = *cprof_seq_ptr(p,j,i);
                if (seq_loc == loc && ! CPROF_IS_BLANK(ch))
                    return j;
                if (! CPROF_IS_BLANK(ch) && CPROF_LETTER_NO(ch) > 0)
                    ++seq_loc;
            }
        return -1;
    }
    /* failed */
}

int cprof_get_seq_first_location(cprof p, identifier_t id, int pos)
{
    int i;
    int best_loc = MAXINT;

    for(i=0; i<p->width; i++)
        if (p->id[i] == id && p->pos[i] > pos && p->pos[i] < best_loc)
            best_loc = p->pos[i];
    return best_loc == MAXINT ? -1 : best_loc;
}

void cprof_count(cprof p, int i, cprof_node N, cprof_node real_N)
{
    unsigned char *s;
    int seq, j;

    for(j=0; j<N_PROFILE_LETTERS; j++)
        real_N[j] = N[j] = 0;
    for(seq=0; seq<p->width; seq++) {
        s = cprof_seq_ptr(p, i, seq);
        if (CPROF_IS_BLANK(*s))
            continue;
        N(CPROF_LETTER_NO(*s))++;
        if (! CPROF_IS_N(*s))
            real_N(CPROF_LETTER_NO(*s))++;
    }
}

```

```

/* Sanity check for profiles */
static void cprof_verify(cprof p)
{
    int i;
    int j;
    int sum;
    int *not_empty;
    cprof_node N, real_N;
    unsigned char ch;

    if ((not_empty=(int*)calloc(p->width, sizeof(int)))==NULL)
        err("cprof_verify: memory failure trying to allocate %d for nodes",
            p->width * sizeof(int));
    for(i=0; i<p->len; i++) {
        sum = 0;
        for(j=0; j<N_PROFILE_LETTERS; j++) {
            assert(p->node[i][j] >= 0);
            sum += p->node[i][j];
        }
        assert(sum > 0);
        cprof_count(p, i, N, real_N);
        /* Possible problem with verifying position 0 (gaps)... */
        for(j=1; j<N_PROFILE_LETTERS; j++)
            assert(N[j] == p->node[i][j] && N[j] >= real_N[j]);
        for(j=0; j<p->width; j++) {
            ch = *cprof_seq_ptr(p, i, j);
            assert((ch & CPROF_LETTER) || ch == CPROF_BLANK);
            if (! CPROF_IS_BLANK(ch))
                not_empty[j]=1;
        }
    }
    for(j=0; j<p->width; j++) { /* Could test more about ids and positions */
        assert(p->id[j] >= 0);
        assert(p->pos[j] >= 0);
        assert(not_empty[j]);
    }
    free(not_empty);
}

/* Create an empty counted profile */
cprof cprof_new(unsigned int n)
{
    cprof p;
    int i, j;

    /* allocate */
    if ((p = malloc(sizeof(struct cprof_s)))
        == NULL)
        return NULL;
    if ((p->node = malloc(n * sizeof(cprof_node)))
        == NULL)
        return NULL;
    if ((p->realnode = malloc(n * sizeof(cprof_node)))
        == NULL)
        return NULL;
    if ((p->link = malloc((n+1) * sizeof(cprof_link)))
        == NULL)
        return NULL;
    if (p)
        free(p);
    if (p->node)
        free(p->node);
    if (p->realnode)
        free(p->realnode);
    return NULL;
}

```

cprof.c

```

}

p->len = n;
p->width = 0;
p->seq = NULL;
p->id = NULL;
p->pos = NULL;
p->very_clean = NULL;

/* initialise */
for(i=0; i<n; i++) {
    for(j=0; j<N_PROFILE_LETTERS; j++)
        p->realnode[i][j] = p->node[i][j] = 0;
    p->link[i] = 0;
}
p->link[n] = 0;
return p;
}

/* Initialise a counted profile according to a (string) sequence */
/* How should composite letters (M,G,R,S,V,W,Y,H,K,D,B,N) be treated?
   What we do here is randomise one of the possible letters, and keep
   the link with weight=1. Is this wise? */
cprof cprof_init(char str[], identifier_t id, short very_clean)
{
    cprof ret;
    int i, j, sum;
    static const char *letters[N_PROFILE_LETTERS] = {
        "", /* gap letters */
        "AMRWDRHVN", /* letters corresponding to 'A' */
        "CMSYBHRVN", /* letters corresponding to 'C' */
        "GKRSBDHVN", /* letters corresponding to 'G' */
        "TKWYBDHVN" /* letters corresponding to 'T' */
    };

    if ((ret = cprof_new(strlen(str))) == NULL)
        return NULL;
    if ((ret->seq = (unsigned char*) malloc(ret->len)) == NULL) {
        free(ret);
        return NULL;
    }
    ret->width = 1;

    i = 0;
    while (str[i]) {
        ret->link[i] = 1; /* Width is 1 */
        sum = 0;
        for(j=1; j<N_PROFILE_LETTERS; j++)
            if (toupper(str[i]) == bases[j])
                ret->realnode[i][j] = 1;
        for(j=0; j<N_PROFILE_LETTERS; j++)
            if (strchr(letters[j], toupper(str[i])) != NULL) {
                ret->node[i][j] = 1; /* Set all matching letters */
                sum++;
            }
        r = rand() % sum;
    }
}

```

cprof.c

```

for(j=0; j<N_PROFILE_LETTERS; j++)
    if (ret->node[i][j]) {
        if (r--)
            ret->node[i][j] = 0;
        else
            *cprof_seq_ptr(ret, i, 0) = j | CPROF_LETTER |
                (sum > 1 ? CPROF_N : 0);
    }
    ++i;
}

/* Set sequence id (this will be propagated for ever and EVER AND EVER) */
if ((ret->id = malloc(sizeof(identifier_t))) == NULL) {
    free(ret->id);
    free(ret->seq);
    free(ret);
    return NULL;
}
if ((ret->pos = malloc(sizeof(int))) == NULL) {
    free(ret->id);
    free(ret->seq);
    free(ret);
    return NULL;
}
if ((ret->very_clean = malloc(sizeof(short))) == NULL) {
    free(ret->id);
    free(ret->seq);
    free(ret->pos);
    free(ret);
    return NULL;
}
*ret->id = id;
*ret->pos = 0;
*ret->very_clean = very_clean;

ret->link[i] = 1; /* Set last link */

CHECK(ret);
return ret;
}

/*
 * Initialise a cprof with dirty ends.
 * Clean area is from start to end (inclusive).
 */
cprof cprof_init_dirty(char str[], identifier_t id, int start, int end)
{
    cprof ret = cprof_init(str, id, 0);
    int i;
    for(i=0; i<start; i++)
        *cprof_seq_ptr(ret, i, 0) |= CPROF_DIRTY;
    for(i=end+1; i<ret->len; i++)
        *cprof_seq_ptr(ret, i, 0) |= CPROF_DIRTY;
    return ret;
}

```

cprof.c

```

/* Free a profile */
void cprof_free(cprof prof)
{
    int i;

    CHECK(prof);
    free(prof->node);
    free(prof->realnode);
    free(prof->link);
    free(prof->seq);
    free(prof->id);
    free(prof->pos);
    free(prof->very_clean);
    free(prof);
}

static void eliminate_squished_seqs(cprof p)
{
    int i, j;
    for(j=p->width-1; j>=0; --j)
        if (p->pos[j] == -1) {
            --p->width;
            memmove(p->pos+j, p->pos+j+1, (p->width-j)*sizeof(int));
            memmove(p->id+j, p->id+j+1, (p->width-j)*sizeof(identifier_t));
            memmove(p->very_clean+j, p->very_clean+j+1, (p->width-j)*sizeof(short));
            memmove(p->seq+j*p->len, p->seq+(j+1)*p->len, (p->width-j)*p->len);
        }
    p->pos = realloc(p->pos, p->width*sizeof(int));
    p->id = realloc(p->id, p->width*sizeof(identifier_t));
    p->very_clean = realloc(p->very_clean, p->width*sizeof(short));
    p->seq = realloc(p->seq, p->width*p->len);
}

/*
 * cprof_update -- Unify a profile into another
 *
 * This is an asymmetrical routine: *all* of p2 is unified with p1.
 * Starting at the given position start. p2 must end before p1.
 * Unification proceeds according to the instructions in str. Since
 * str refers to a possibly different order of the 2 profiles, the
 * flag flip allows reversing the meaning.
 */
void cprof_update(cprof p1, cprof p2, int start1, int start2,
    char *str, int flip)
{
    int i, i1, i2, j, k;
    cprof_node *node;
    cprof_node *realnode;
    cprof_link *link;
    char s;
    unsigned char *seq;
    unsigned char ch;
    int len;
    int *started;
    int *ended;

    /* Array of indicators if sequences started */
    /* Array of indicators if sequences ended */
}

```

cprof.c

```

int w1 = p1->width, w2 = p2->width;
extern int improve_cprof(cprof);
CHECK(p1);
CHECK(p2);

/* Calculate length of new profile, allocate memory */
for(i=0; i<start1; i++)
    if ((str[i] == ALIGN_GAP1 && !flip) ||
        (str[i] == ALIGN_GAP2 && flip))
        len++;
if ((node = malloc(len * sizeof(cprof_node))) == NULL)
    err("cprof_update: memory failure trying to allocate %d for nodes",
        len * sizeof(cprof_node));
if ((realnode = malloc(len * sizeof(cprof_node))) == NULL)
    err("cprof_update: memory failure trying to allocate %d for realnodes",
        len * sizeof(cprof_node));
if ((link = malloc((len+1) * sizeof(cprof_link))) == NULL)
    err("cprof_update: memory failure trying to allocate %d for links",
        (len+1) * sizeof(cprof_link));
if ((seq = (unsigned char *)malloc((w1+w2) * len)) == NULL)
    err("cprof_update: memory failure trying to allocate %dx%d for alignment",
        w1+w2, len);
if ((started = (int *)malloc((w1+w2)*sizeof(int))) == NULL ||
    (ended = (int *)malloc((w1+w2)*sizeof(int))) == NULL)
    err("cprof_update: memory failure trying to allocate %d for flags",
        w1+w2);
for(j=0; j<w1; j++) {
    started[j] = *cprof_seq_ptr(p1, start1, j) != ' ';
    ended[j] = 0;
}
for(j=0; j<w2; j++) {
    started[j+w1] = *cprof_seq_ptr(p2, start2, j) != ' ';
    ended[j+w1] = 0;
}

/* Copy initial segment of p1 */
for(i=0; i<start1; i++) {
    link[i] = p1->link[i];
    for(j=0; j<N_PROFILE_LETTERS; j++) {
        realnode[i][j] = p1->realnode[i][j];
        node[i][j] = p1->node[i][j];
    }
}
for(j=0; j<w1; j++) {
    ch = *cprof_seq_ptr(p1, i, j);
    seq[j * len + i] = ch;
    if (! CPROF_IS_BLANK(ch))
        started[j] = 1;
    else if (started[j])
        ended[j] = 1;
}
for(j=0; j<w2; j++)
    seq[(j+w1) * len + i] = CPROF_BLANK;

```

cprof.c

```

)
/* Merge p1 and p2 */
for(i=start1, i2=start2, i2=start1; i++ < link[i] = p1->link[i1] + p2->link[i2];
    s = str[i-start1];
    if (!flip)
        if (s == ALIGN_GAP2)
            s = ALIGN_GAP1;
        else if (s == ALIGN_GAP1)
            s = ALIGN_GAP2;
    switch(s) {
    case ALIGN_MATCH:
        for(j=0; j<N_PROFILE_LETTERS; j++) {
            realnode[i1][j] = p1->realnode[i1][j] + p2->realnode[i2][j];
            node[i1][j] = p1->node[i1][j] + p2->node[i2][j];
        }
        for(j=0; j<w1; j++)
            seq[j * len + i1] = *cprof_seq_ptr(p1, i1, j);
        for(j=0; j<w2; j++)
            seq[(j+w1) * len + i] = *cprof_seq_ptr(p2, i2, j);
        i1++; i2++;
        break;
    case ALIGN_GAP1:
        for(j=0; j<N_PROFILE_LETTERS; j++) {
            realnode[i1][j] = p2->realnode[i2][j];
            node[i1][j] = p2->node[i2][j];
        }
        realnode[i1][0] += p1->link[i1];
        node[i1][0] += p1->link[i1];
        for(j=0; j<w1; j++)
            seq[j * len + i] = (started[j] && !ended[j]) ?
                (CPROF_LETTER | (CPROF_IS_DIRTY(seq[j*len+i-1]) ? CPROF_DIRTY : 0)) :
                CPROF_BLANK;
        for(j=0; j<w2; j++)
            seq[(j+w1) * len + i] = *cprof_seq_ptr(p2, i2, j);
        i2++;
        break;
    case ALIGN_GAP2:
        for(j=0; j<N_PROFILE_LETTERS; j++) {
            realnode[i1][j] = p1->realnode[i1][j];
            node[i1][j] = p1->node[i1][j];
        }
        realnode[i1][0] += p2->link[i2];
        node[i1][0] += p2->link[i2];
        for(j=0; j<w1; j++)
            seq[j * len + i] = *cprof_seq_ptr(p1, i1, j);
        for(j=0; j<w2; j++)
            seq[(j+w1) * len + i] = (started[j+w1] && !ended[j+w1]) ?
                (CPROF_LETTER | (CPROF_IS_DIRTY(seq[j*len+i-1]) ? CPROF_DIRTY : 0)) :
                CPROF_BLANK;

```

cprof.c

```

        CPprof_BLANK;
        il++;
        break;
    default:
        err("Unknown alignment character '%c' (%d) in cprof_update",
            s, s);
        break;
    }
    if (s != ALIGN_GAP2) /* UNREACHED */
        break;
    if (s != ALIGN_GAP2) /* i2 changed */
        for (j=0; j<w2; j++) {
            if (i1 < p1->len)
                started[j+w1] = ! CPprof_IS_BLANK(*cprof_seq_ptr(p2,i2,j));
            if (started[j+w1] && i2 < p2->len)
                ended[j+w1] = CPprof_IS_BLANK(*cprof_seq_ptr(p2,i2,j));
        }
    if (s != ALIGN_GAP1) /* i1 changed */
        for (j=0; j<w1; j++) {
            if (i1 < p1->len)
                started[j] = ! CPprof_IS_BLANK(*cprof_seq_ptr(p1,i1,j));
            if (started[j] && i1 < p1->len)
                ended[j] = CPprof_IS_BLANK(*cprof_seq_ptr(p1,i1,j));
        }
    link[i1] = p1->link[i1] + p2->link[i2];
    assert(i1 <= p1->len && i2 <= p2->len);

    /* Copy remainder of p1 */
    for (; i1 < p1->len; i1++, i++) {
        for (j=0; j<N_PROFILE_LETTERS; j++) {
            realnode[i1][j] = p1->realnode[i1][j];
        }
        node[i1][j] = p1->node[i1][j];
        link[i1+1] = p1->link[i1+1];
        for (j=0; j<w1; j++)
            seq[j * len + i1] = *cprof_seq_ptr(p1, i1, j);
        for (j=0; j<w2; j++)
            seq[(j*w1) * len + i1] = CPprof_BLANK;
    }

    /* Merge p2's sequences' identification with p1's */
    if ((p1->id = realloc(p1->id, (w1+w2)*sizeof(identifier_t))) == NULL)
        err("Memory failure in cprof_update for %d ids", w1+w2);
    if ((p1->pos = realloc(p1->pos, (w1+w2)*sizeof(int))) == NULL)
        err("Memory failure in cprof_update for %d positions", w1+w2);
    if ((p1->very_clean = realloc(p1->very_clean, (w1+w2)*sizeof(short)))
        == NULL)
        err("Memory failure in cprof_update for %d positions", w1+w2);

    for (j=0; j<w2; j++) {
        for (k=start2; k<i2; k++)
            if (! CPprof_IS_BLANK(*cprof_seq_ptr(p2,k,j)))
                break;
        p1->id[w1+j] = p2->id[j]; /* Empty row -- mark for squishing */
        if (k == i2)
            p1->pos[w1+j] = -1;
        else {
            /* Identify location (in seq) of start2 */

```

```

        int loc;
        for (k=0, loc=p2->pos[j]; k<start2; k++) {
            ch = *cprof_seq_ptr(p2,k,j);
            if (! CPprof_IS_BLANK(ch) && CPprof_LETTER_NO(ch) > 0)
                loc++;
        }
        p1->pos[w1+j] = loc;
        p1->very_clean[w1+j] = p2->very_clean[j];
    }

    /* Free (most of) p1; return the result in p1 */
    /* cprof_free(p2); DON'T FREE p2 (avner) */
    free(p1->node);
    free(p1->link);
    free(p1->seq);
    p1->node = node;
    p1->realnode = realnode;
    p1->link = link;
    p1->seq = seq;
    p1->len = len;
    p1->width = w1 + w2;

    free(started);
    free(ended);

    eliminate_squished_seqs(p1);
    (void)improve_cprof(p1);

#ifdef DUMP_CPROF
    cprof_dump(p1); /* *** DBG *** */
#endif
    CHECK(p1);
}

/* * cut off head of prof. The head is lost (and memory reclaimed) */
void cprof_cut_head(cprof p, int offset)
{
    int node_sz = (p->len - offset)*sizeof(cprof_node);
    int link_sz = (p->len-1 - offset)*sizeof(cprof_link);
    int i, j;
    unsigned char ch;

    CHECK(p);

    assert(offset < p->len);

    memmove(p->node, p->node+offset, node_sz);
    memmove(p->realnode, p->realnode+offset, node_sz);
    memmove(p->link, p->link+offset, link_sz);

    /* Eliminate from the multiple alignment any sequences which will

```

```

be completely snipped */
/* First scan and mark all sequences which will vanish */
for(j=0; j<p->width; j++) {
    short nothing_left = 1;
    /* Scan to see if alignment has any letters */
    for(i=offset; i<p->len; i++) {
        ch = *cprof_seq_ptr(p,i,j);
        if (! CPROF_IS_BLANK(ch))
            nothing_left = 0;
    }
    if (nothing_left)
        p->pos[j] = -1;
}
eliminate_squished_seqs(p);

/* Correct starting positions (ids are unchanged) */
for(j=0; j<p->width; j++)
    for(i=0; i<offset; i++) {
        ch = *cprof_seq_ptr(p,i,j);
        if (! CPROF_IS_BLANK(ch) && CPROF_LETTER_NO(ch) > 0)
            p->pos[j]++;
    }

/* Re-shape p->seq array */
for(j=0; j<p->width; j++)
    for(i=offset; i<p->len; i++)
        p->seq[j*(p->len-offset) + i-offset] = p->seq[j*p->len + i];

if ((p->node = realloc(p->node, node_sz)) == NULL)
    err("cprof_cut_head: memory failure trying to allocate %d for nodes",
        node_sz);
if ((p->realnode = realloc(p->realnode, node_sz)) == NULL)
    err("cprof_cut_head: memory failure trying to allocate %d for realnodes",
        node_sz);
if ((p->link = realloc(p->link, link_sz)) == NULL)
    err("cprof_cut_head: memory failure trying to allocate %d for links",
        link_sz);
if ((p->seq = realloc(p->seq, p->width*(p->len-offset))) == NULL)
    err("cprof_cut_head:
        "memory failure trying to allocate %dx%d for alignment",
        p->width, p->len-offset);
p->len -= offset;
CHECK(p);
}

/* * cut off tail of prof. The tail is lost (and memory reclaimed)
*/
void cprof_cut_tail(cprof p, int offset)
{
    int node_sz = (p->len - offset)*sizeof(cprof_node);
    int link_sz = (p->len+1-offset)*sizeof(cprof_link);
    int i, j;

```

cprof.c

```

unsigned char ch;
CHECK(p);
assert(offset < p->len);
/* Don't need to move p->node, p->link updated contents in memory */
/* Eliminate from the multiple alignment any sequences which will
be completely snipped */
for(j=0; j<p->width; j++) {
    short nothing_left = 1;
    /* Scan to see if the alignment has any letters */
    for(i=0; i<p->len-offset; i++) {
        ch = *cprof_seq_ptr(p,i,j);
        if (! CPROF_IS_BLANK(ch))
            nothing_left = 0;
    }
    if (nothing_left)
        p->pos[j] = -1;
}
eliminate_squished_seqs(p);

/* No need to correct starting positions (and ids are unchanged) */

/* Re-shape p->seq array */
for(j=0; j<p->width; j++)
    for(i=0; i<p->len-offset; i++)
        p->seq[j*(p->len-offset) + i] = p->seq[j*p->len + i];

if ((p->node = realloc(p->node, node_sz)) == NULL)
    err("cprof_cut_tail: memory failure trying to allocate %d for nodes",
        node_sz);
if ((p->realnode = realloc(p->realnode, node_sz)) == NULL)
    err("cprof_cut_tail: memory failure trying to allocate %d for realnodes",
        node_sz);
if ((p->link = realloc(p->link, link_sz)) == NULL)
    err("cprof_cut_tail: memory failure trying to allocate %d for links",
        link_sz);
if ((p->seq = realloc(p->seq, p->width * (p->len-offset))) == NULL)
    err("cprof_cut_tail:
        "memory failure trying to allocate %dx%d for alignment",
        p->width, p->len-offset);
p->len -= offset;
CHECK(p);
}

/* * split prof into tail (returned) and head (prof is converted to head)
*/
cprof cprof_split(cprof p, int offset)
{
    cprof tail;
    int tail_node_sz = (p->len - offset)*sizeof(cprof_node);

```

cprof.c

```

int tail_link_sz = (p->len+1 - offset)*sizeof(cprof_link);
int head_node_sz = offset * sizeof(cprof_node);
int head_link_sz = (offset+1)*sizeof(cprof_link);
int i, j;

CHECK(p);

assert(0 < offset && offset < p->len);

tail = cprof_segment(p, offset, p->len-1);

cpref_cut_tail(p, p->len-offset);

CHECK(p);
CHECK(tail);

return tail;
}

/*
 * return a (new) copy of a profile
 */
cpref cprof_dup(cprof p)
{
    cprof new;

    CHECK(p);

    if ((new = cprof_new(p->len)) == NULL)
        return NULL;

    new->width = p->width;
    if ((new->seq = malloc(new->width * new->len)) == NULL)
        err("cpref_dup: memory failure trying to allocate %dx%d for alignment",
            new->width, new->len);
    if ((new->id = (identifier_t *)
        malloc(new->width * sizeof(identifier_t))) == NULL)
        err("cpref_dup: memory failure trying to allocate %d for ids",
            new->width);
    if ((new->pos = (int *) malloc(new->width * sizeof(int))) == NULL)
        err("cpref_dup: memory failure trying to allocate %d for pos's",
            new->width);
    if ((new->very_clean = (short *) malloc(new->width * sizeof(short))) == NULL)
        err("cpref_dup: memory failure trying to allocate %d for cleanliness",
            new->width);

    memcpy(new->node, p->node, p->len * sizeof(cprof_node));
    memcpy(new->realnode, p->realnode, p->len * sizeof(cprof_node));
    memcpy(new->link, p->link, (p->len+1) * sizeof(cprof_link));
    memcpy(new->seq, p->seq, p->width * p->len);
    memcpy(new->id, p->id, p->width * sizeof(identifier_t));
    memcpy(new->pos, p->pos, p->width * sizeof(int));
    memcpy(new->very_clean, p->very_clean, p->width * sizeof(short));

    CHECK(new);

    return new;
}

```

```

/*
 * cprof_glue -- glue 2 profiles one after the other.
 *
 * The link connecting the 2 profiles has the weight of the relevant
 * link of the _secondary_profile. The primary profile grows, while
 * the secondary profile is freed. The flag says whether the
 * secondary profile should come before or after the primary.
 */

/* *** DOESN'T GLUE CONSECUTIVE IDS!!! *** */

void cprof_glue(cprof primary, cprof secondary,
               enum glue_mode flag, int link)
{
    int node_sz = (primary->len+secondary->len) * sizeof(cprof_node);
    int link_sz = (primary->len+secondary->len+1) * sizeof(cprof_link);
    unsigned char *seq;
    int i, j;

    CHECK(primary);
    CHECK(secondary);

    if ((seq = malloc((primary->width+secondary->width) *
        (primary->len+secondary->len))) == NULL)
        err("cpref_glue: memory failure allocating %dx%d for alignment",
            primary->width + secondary->width, primary->len + secondary->len);

    switch(flag) {
        case e_before:
            /* secondary, then primary */
            if ((secondary->node = realloc(secondary->node, node_sz)) == NULL)
                err("cpref_glue: memory failure allocating %d for secondary->node",
                    node_sz);
            if ((secondary->realnode = realloc(secondary->realnode, node_sz)) == NULL)
                err("cpref_glue: memory failure allocating %d for secondary->realnode",
                    node_sz);
            if ((secondary->link = realloc(secondary->link, link_sz)) == NULL)
                err("cpref_glue: memory failure allocating %d for secondary->link",
                    link_sz);
            memcpy(secondary->node+secondary->len, primary->node,
                primary->len*sizeof(cprof_node));
            memcpy(secondary->realnode+secondary->len, primary->realnode,
                primary->len*sizeof(cprof_node));
            memcpy(secondary->link+secondary->len+1, primary->link,
                primary->len*sizeof(cprof_link));
            secondary->link[secondary->len] = link;
            for(i=0; i<secondary->len+primary->len; i++)
                seq[i] * (primary->len+secondary->len) + i] =
                    i < secondary->len ? cprof_seq_ptr(secondary, i, j) : CPROF_BLANK;
            for(i=0; i<secondary->len+primary->len; i++)
                for(j=0; j<primary->width; j++)
                    seq[(j+secondary->width) * (primary->len+secondary->len) + i] =
                        i < secondary->len ? CPROF_BLANK :
                            *cpref_seq_ptr(primary, i-secondary->len, j);

            if ((secondary->id = (identifier_t *)
                realloc(secondary->id,

```



```

        (primary->width*secondary->width)*sizeof(identifier_t)))
    == NULL)
    err("cprof_glue: memory failure allocating %d for secondary->id",
        primary->width*secondary->width);
    if ((secondary->pos = (int *)
        realloc(secondary->pos,
            (primary->width*secondary->width)*sizeof(int)))
        == NULL)
        err("cprof_glue: memory failure allocating %d for secondary->pos",
            primary->width*secondary->width);
    if ((secondary->very_clean = (short *)
        realloc(secondary->very_clean,
            (primary->width*secondary->width)*sizeof(short)))
        == NULL)
        err("cprof_glue: memory failure allocating %d for secondary->very_clean",
            primary->width*secondary->width);
    memcpy(secondary->id + secondary->width,
        primary->id,
        primary->pos);
    memcpy(secondary->pos+secondary->width,
        primary->pos,
        primary->pos);
    memcpy(secondary->very_clean+secondary->width,
        primary->very_clean,
        primary->very_clean);
    free(primary->node);
    free(primary->realnode);
    free(primary->link);
    free(primary->seq);
    free(secondary->seq);
    free(primary->id);
    free(primary->pos);
    free(primary->very_clean);
    primary->width += secondary->width;
    primary->node = secondary->node;
    primary->realnode = secondary->realnode;
    primary->link = secondary->link;
    primary->seq = seq;
    primary->len += secondary->len;
    primary->id = secondary->id;
    primary->pos = secondary->pos;
    primary->very_clean = secondary->very_clean;
    free(secondary);
    break;
}

case e_after:
    /* primary, then secondary */
    if ((primary->node = realloc(primary->node, node_sz)) == NULL)
        err("cprof_glue: memory failure allocating %d for primary->node",
            node_sz);
    if ((primary->realnode = realloc(primary->realnode, node_sz)) == NULL)
        err("cprof_glue: memory failure allocating %d for primary->realnode",
            node_sz);
    if ((primary->link = realloc(primary->link, link_sz)) == NULL)
        err("cprof_glue: memory failure allocating %d for primary->link",
            link_sz);
    memcpy(primary->node+primary->len, secondary->node,
        secondary->len*sizeof(cprof_node));
    memcpy(primary->realnode+primary->len, secondary->realnode,
        secondary->len*sizeof(cprof_node));

```

cprof.c

```

    memcpy(primary->link+primary->len+1, secondary->link+1,
        secondary->len*sizeof(cprof_link));
    primary->link[primary->len] = link;
    for(i=0; i<primary->len+secondary->len; i++)
        for(j=0; j<primary->width; j++)
            seq[j] = (primary->len+secondary->len) + i = CPROF_BLANK;
    for(i=0; i<primary->len+secondary->len; i++)
        for(j=0; j<secondary->width; j++)
            seq[(j*primary->width) + (primary->len+secondary->len) + i] =
                i < primary->len ? CPROF_BLANK :
                    *cprof_seq_ptr(secondary, i-primary->len, j);
    if ((primary->id = (identifier_t *)
        realloc(primary->id,
            (primary->width*secondary->width)*sizeof(identifier_t)))
        == NULL)
        err("cprof_glue: memory failure allocating %d for primary->id",
            primary->width*secondary->width);
    if ((primary->pos = (int *)
        realloc(primary->pos,
            (primary->width*secondary->width)*sizeof(int)))
        == NULL)
        err("cprof_glue: memory failure allocating %d for primary->pos",
            primary->width*secondary->width);
    if ((primary->very_clean = (short *)
        realloc(primary->very_clean,
            (primary->width*secondary->width)*sizeof(short)))
        == NULL)
        err("cprof_glue: memory failure allocating %d for primary->very_clean",
            primary->width*secondary->width);
    memcpy(primary->id + primary->width,
        secondary->id,
        secondary->pos);
    memcpy(primary->pos+primary->width,
        secondary->pos,
        secondary->pos);
    memcpy(primary->very_clean+primary->width,
        secondary->very_clean,
        secondary->very_clean);
    primary->len += secondary->len;
    primary->width += secondary->width;
    free(primary->seq);
    primary->seq = seq;
    cprof_free(secondary);
    break;
}

default:
    err("cprof_glue: unknown glue mode %d", flag);
    break;
    /* UNREACHED */
}

CHECK(primary);
}

/* Return a new profile consisting of prof from start to end (inclusive) */
cprof_cprof_segment(cprof_prof, int start, int end)
{

```

cprof.c

```

int len = (end-start)+1; /* ??? support reverse ??? */
cprof ret;
int i, j;
unsigned char ch;

CHECK(prof);

assert(0 <= start && start <= end && end < prof->len);

if ((ret = cprof_new(len)) == NULL)
    err("cprof_segment: memory failure allocating %d nodes for ret",
        len);

ret->width = prof->width;
if ((ret->seq = malloc(ret->width*ret->len)) == NULL)
    err("cprof_segment: memory failure allocating %dx%d for alignment",
        ret->width + ret->len);

if ((ret->id = (identifier_t *)
    malloc(ret->width*sizeof(identifier_t))) == NULL)
    err("cprof_segment: memory failure allocating %d for ids", ret->width);
if ((ret->pos = (int *) malloc(ret->width*sizeof(int))) == NULL)
    err("cprof_segment: memory failure allocating %d for pos", ret->width);
if ((ret->very_clean = (short *) malloc(ret->width*sizeof(short))) == NULL)
    err("cprof_segment: memory failure allocating %d for very_clean",
        ret->width);

memcpy(ret->id, prof->id, ret->width*sizeof(identifier_t));
memcpy(ret->pos, prof->pos, ret->width*sizeof(int));
memcpy(ret->very_clean, prof->very_clean, ret->width*sizeof(short));

/* Update sequence positions */
for(j=0; j<prof->width; j++)
    for(i=0; i<start; i++) {
        ch = *cprof_seq_ptr(prof, i, j);
        if (! CPROF_IS_BLANK(ch) && CPROF_LETTER_NO(ch) > 0)
            ++ret->pos[j];
    }

for(j=0; j<prof->width; j++) {
    short nothing_left = 1;
    for(i=start; i<=end; i++) {
        ch = *cprof_seq_ptr(ret, i-start, j) = *cprof_seq_ptr(prof, i, j);
        /* Check to see if alignment has any letters */
        if (! CPROF_IS_BLANK(ch))
            nothing_left = 0;
    }
    if (nothing_left)
        ret->pos[j] = -1;
}

memcpy(ret->node, prof->node+start, len*sizeof(cprof_node));
memcpy(ret->realnode, prof->realnode+start, len*sizeof(cprof_node));
memcpy(ret->link, prof->link+start, (len+1)*sizeof(cprof_link));

eliminate_squished_seqs(ret);

CHECK(ret);

```

cprof.c

```

return ret;
}

void cprof_dump(cprof prof)
{
    int i, j, k;
    unsigned char ch;

    for(i=0; i<(prof->len+59)/60; i++) {
        /* sequence */
        printf("%4d ", 60*i+1);
        for(j=60*i; j<prof->len && j< 60*(i+1); j++)
            putchar(cprof_node_char(prof, j, NULL, NULL));
        printf(" %4d\n", 60*(i+1));
        /* alignment */
        for(k=0; k<prof->width; k++) {
            printf("%4d ", prof->id[k]);
            for(j=60*i; j<prof->len && j<60*(i+1); j++) {
                ch = *cprof_seq_ptr(prof, j, k);
                putchar(CPROF_IS_BLANK(ch) ? ' ' :
                    CPROF_IS_N(ch) ? 'N' :
                    CPROF_IS_DIRTY(ch) ? tolower(bases[CPROF_LETTER_NO(ch)]) :
                    bases[CPROF_LETTER_NO(ch)]);
            }
            putchar('\n');
        }
    }

    int cprof_first_clean(cprof p)
    {
        int j, width, dirty, start;
        unsigned char ch;

        for(start=0; start<p->len; start++) {
            width=0;
            dirty=0;
            for(j=0; j<p->width; j++) {
                ch = *cprof_seq_ptr(p, start, j);
                if (! CPROF_IS_BLANK(ch))
                    width++;
                if (CPROF_IS_DIRTY(ch))
                    dirty++;
            }
            if (width > 1 || ! dirty)
                return start;
        }

    int cprof_last_clean(cprof p)
    {
        int j, width, dirty, end;
        unsigned char ch;

        for(end=p->len-1; end>=0; end--) {
            width=0;
            dirty=0;

```

cprof.c

```

for(j=0;j<p->width;j++) {
    ch = *cprof_seq_ptr(p,end,j);
    if (! CPROF_IS_BLANK(ch))
        width++;
    if(CPROF_IS_DIRTY(ch))
        dirty++;
}
if(width > 1 || ! dirty)
    return end;
}

/* Return (non-redundant) list of at most SZ ids from P in IDS; return
 * number of such ids. It is safe to pass SZ=0 (and a valid, e.g. NULL,
 * pointer in IDS) in order to get the correct value of SZ.
 */
static int cmp_id(identifier_t *a, identifier_t *b)
{
    return (int)*a - (int)*b;
}

int cprof_ids(cprof p, identifier_t *ids, int sz)
{
    identifier_t *list;
    int m,n;

    /* Copy list of ids from p */
    if ((list = malloc(p->width * sizeof(identifier_t))) == NULL)
        err("cprof_ids: memory failure for %d", p->width);
    memcpy(list, p->ids, p->width*sizeof(identifier_t));

    /* Sort and uniquify it */
    bsort(list, p->width, sizeof(identifier_t), cmp_id);
    for(m=1,n=1; m<p->width; m++)
        if (list[m] != list[m-1])
            list[n++] = list[m];

    /* Return array and #unique elements */
    if (sz > 0)
        memcpy(ids, list, (sz < n ? sz : n) * sizeof(identifier_t));
    return n;
}

/* Return (non-redundant) list of at most SZ ids which align to POS in P
 * in IDS; return number of such ids.
 * It is safe to pass SZ=0 (and a valid, e.g. NULL, pointer in IDS)
 * in order to get the correct value of SZ.
 */
int cprof_pos_ids(cprof p, int pos, identifier_t *ids, int sz)
{
    identifier_t *list;
    int i,m,n,num=0;

    /* Copy list of ids from p */
    if ((list = malloc(p->width * sizeof(identifier_t))) == NULL)
        err("cprof_ids: memory failure for %d", p->width);
    for(i=0; i<p->width; i++)
        if (! CPROF_IS_BLANK(*cprof_seq_ptr(p,pos,i)))

```

```

list[num++] = p->id[i];
}

/* Sort and uniquify it */
bsort(list, num, sizeof(identifier_t), cmp_id);
for(m=1,n=1; m<num; m++)
    if (list[m] != list[m-1])
        list[n++] = list[m];

/* Return array and #unique elements */
if (sz > 0)
    memcpy(ids, list, (sz < n ? sz : n) * sizeof(identifier_t));
return n;
}

void cprof_fix_id_after_cut(cprof p, int id, int cutpoint)
{
    int i;
    for(i=0; i<p->width; i++) {
        if(p->id[i] == id) {
            assert(p->pos[i] >= cutpoint);
            p->pos[i] -= cutpoint;
        }
    }
}

```

```

/*
 * cprof_align.c -- support routine(s) for cprof_align's
 */
/*
 * ariels@compugen 26/10/97
 */
/*
 * $Log: cprof_align.c,v $
 * Revision 1.2 1997/11/30 07:48:39 ariels
 * Added Changelog comment and static rcsid.
 */
static char rcsid[] = "$Id: cprof_align.c,v 1.2 1997/11/30 07:48:39 ariels Rel $";

#include <stdlib.h>
#include "cprof_align.h"

/* Free a profile alignment (linked list) */
void cprof_align_free(cprof_align align)
{
    cprof_align a;

    while (align != NULL) {
        a = align->next;
        free(align->str);
        free(align);
        align = a;
    }
}

```



```

/*
* $Log: custody_zones.c,v $
* Revision 1.23 1998/07/03 14:02:07 avner
* adding 'rules-usage' message when discarding a small alignment
* (that floats).
*
* Revision 1.22 1998/07/01 06:56:08 eyal
* Fix bug in czm_to_node_path
*
* Revision 1.21 1998/06/29 12:09:09 eyal
* Shift prototypes to .h file (where they belong).
*
* Revision 1.20 1998/06/28 13:45:57 eyal
* Adding function for node_path_mode
*
* Revision 1.19 1998/06/18 14:42:47 ariels
* Strip "#ifdef CPROF_ALIGN" and "#ifdef IMPROVE_CPROF" lines.
*
* Revision 1.18 1998/04/29 10:33:45 eyal
* (done by avner) Fix bug caused by 'resolve_medium_intersection'. Alignment
* could have been cut to a unlimited small size, without a check. A check was n
*
* added, and a new flag in align_data structure was added 'marked_cancelled'. A
*
* interactive_mode for determining which alignment to through is handled better
* this way.
*
* Revision 1.17 1998/04/23 16:41:32 eyal
* correct possible bug in unpure_seglen
*
* Revision 1.16 1998/04/15 09:24:04 ariels
* Correctly call the new cprof_node_char().
*
* Revision 1.15 1998/04/09 11:11:23 ariels
* Merge (copy) branch 1.14.1 with trunk.
*
* Revision 1.14.1.1 1998/02/23 14:30:30 ariels
* Call cprof_node_char with the extra argument mandated by the klugey 'N'-
* counting code.
*
* Revision 1.14 1998/02/22 17:21:50 ariels
* Use a REAL '%' sign in format strings.
* Use new error and warning reporting facility.
* Eliminate problem_msg[]..
*
* Revision 1.13 1998/02/22 11:10:37 avner
* Make 'big intersection' messages more informative. Include the information
* about the similarity between the alignments. A first step towards
* discriminating alignments when possible.
*
* Revision 1.12 1998/02/19 18:52:40 avner
* Fixed definition of 'update_replace_czm_of_same_node' when CPROF_ALIGN not
* defined. Needed to get extra parameter 'splice_graph'.
*
* Revision 1.11 1998/02/17 16:15:09 ariels
* Fix name of global variable cluster_no (to contig_name), since it was
* neither CLUSTER nor NO (number)...
*
* Revision 1.10 1998/02/15 08:36:38 eyal
* Adding function unpure_seglen.

```

```

* Revision 1.9 1998/02/14 17:25:32 avner
* Fix length calculations of tails of alignment, for the sake of concise
* handling of 'almost surely gaps' in locations of the profile ('X').
* Introduce function 'pure_seglen' for that matter.
*
* Revision 1.8 1998/02/11 15:25:38 avner
* stop using ALIGN_MISTAKE_THRESHOLD and use 'indep_node_len' instead.
*
* Revision 1.7 1998/02/01 22:11:58 avner
* Fixing a bug in 'czm_shorten_start'. ALIGN_GAP1 and ALIGN_GAP2 were
* used in a mixed up manner, which caused wrong length calculations.
*
* Revision 1.6 1998/01/21 07:52:35 ariels
* Put err() in scope of prototype (add #include "error.h")
*
* Revision 1.5 1998/01/11 14:20:25 eyal
* Adding support for ident and similarity percent in align_data
*
* Revision 1.4 1998/01/02 13:22:23 avner
* negligible printouts changes.
*
* Revision 1.3 1997/12/22 13:00:18 eyal
* Closing '' in the head documentation - a nasty bug which prevent hilit
* from coloring the code correctly!
*
* Revision 1.2 1997/11/30 07:50:48 ariels
* Added ChangeLog comment and static rcsid.
*
*
static char rcsid[] = "$Id: custody_zones.c,v 1.23 1998/07/03 14:02:07 avner Exp $";

#include "cpof_align.h"
#include "splice_graph.h"
#include "custody_zones.h"
#include "parameters.h"
#include "error.h"
#include <assert.h>

extern indep_node_len, algn_intersec_low, algn_intersec_high;
extern char p_alignstr, interactive;
extern int local_olap_mode;

static int range_for_gluing(int len);
static int compare_alignments(align_data *first, align_data *second);
static void align_data_free(align_data *p);
void align_data_print(align_data *ad);
static void align_data_copy(align_data *from, align_data *to);
static void intersection_message(align_data *ad, align_data *ad2);
static int is_gap_symbol(char c, int which);
static int next_real_node(align_data *p);
void align_data_split(align_data *ad, int cutpoint, cprof_p, int which);
cpof_node_path_to_cprof(path_node *node_path, splice_graph *graph);
void break_path_ad(align_data **path_ad, cprof_path_cprof,
                  path_node *node_path, splice_graph *graph);
path_node *path_node_new(int node_id);

```

```
void path_node___print(path_node *pn);
void path_node___free(path_node *pn);
void path_node_list___free(path_node *pn);

/* the 'subject' structure here is align_data, but we prefer to preserve
 * the different meanings of the align_data used in the hilltops phase,
 * (which will be called 'ad' generally), and the one here, which will
 * usually be called 'czm'

 * a czm is a function of an entering sequence/profile and a splice-graph.
 * it is a partition of the sequence to segments, where each segment is
 * either a best alignment with a sequence of a node of the splice-graph,
 * or it is 'free', meaning it is one of the segments remaining for
 * the partition of the sequence, after the segments of the first type are
 * being removed.

 * One of the very important questions the algorithm should sort-out is
 * partly handled here. The question is
 * 'is a given alignment between two sequences result from the fact that
 * they contain the same segment (aligned segment) in the rna ('they' =
 * the ests themselves, or indirectly, the data propagating from ests
 * provided the previous stages were correct).
 * We separate the alignments we find into two categories: 'certain' and
 * 'dubious', and at the end of the process all 'certain' remain, but
 * not necessarily all 'dubious'.
 * The criterion we use is the existence of a dubious alignment next to
 * another alignment. This is an heuristic using the small probability
 * of a false positive in this case.

 * Defacto, we might have some non-empty small intersections between
 * non-free segments, but we don't expect big ones if ALL alignments were
 * incorporated in the graph since then at no stage information is kept
 * in more than one copy. It can be, however, that small intersections occur
 * and to resolve these, we shorten one or both alignments while considering
 * the quality of the alignments in those sites.
 * quality of the hide parts of the alignment in
 */
```

```
/* logic functions */
/*=====
/*=====
align_data *adl_to_czm(align_data *adl,int len_est,cprof entering_profile)
/*
 * adl_to_czm :: (logic func)
 * given an est and the linked list of the alignments it has with other
 * nodes, create a similar list with the following changes:
 * (i) remove all 'dubious' zones, that is short alignments, that
 * do not satisfy the special condition needed for it to survive (being
 * close to a 'certain' zone).
 * (ii) make the list a full partition of the est, by adding a 'FREE_SEG'
 * custody zones wherever a segment is not within any alignment
 * a typical picture will look something like
 *
 * 1(c) 2(d) 3(c) 4(d) 5(c)
```

custody_zones.c

```
* adl = -----|-----|-----|-----|-----|-----|-----|
*
*
* czm = 1 2 3 FREE_SEG 5 FREE_SEG
*
*
/*
 * align_data *ad,*czm=NULL,**pczm;
 * align_data *last_free = NULL;
 * int len_intersection_segment,pure_len_intersection_segment;
 * int current_should_survive_post,current_should_survive_pre;
 * next_should_survive,current_stay;
 * current_should_survive_pre=next_should_survive=0;

 * pczm=&czm;

 /* taking care of (possible) first free segment */
 if (adl==NULL)
 {
 *pczm = align_data_new(NULL,0,len_est-1);
 return czm;
 }
 if (adl->start_est>0)
 {
 *pczm = align_data_new(NULL,0,adl->start_est-1);
 last_free=*pczm;
 pczm = &((*pczm)->next);
 }
 for (ad=adl; ad->next!=NULL; ad=ad->next,
 current_should_survive_pre=next_should_survive)
 {
 len_intersection_segment = ad->end_est - (ad->next->start_est);
 if (len_intersection_segment < 0) {
 pure_len_intersection_segment = -pure_seglen(entering_profile,
 ad->end_est+1,
 (ad->next)->start_est-1);
 }
 else {
 pure_len_intersection_segment = pure_seglen(entering_profile,
 (ad->next)->start_est,
 ad->end_est);
 }
 if (pure_len_intersection_segment >= align_intersec_low)
 if (pure_len_intersection_segment < align_intersec_high) {
 resolve_medium_intersection(ad,ad->next,len_est);
 if (pure_seglen(entering_profile,ad->start_est,ad->end_est) <
 HIGH_LEN_BOUND)
 ad->marked_cancelled = 1;
 ad->next->marked_cancelled = 1;
 }
 else { /* we have a BIG intersection */
 char answer;
 intersection_message(ad,ad->next);
 if (interactive=='y') {
 printf("which one to cancel (1/2)?");
 answer = getc(stdin);
 }
 }
 }
 }
```

custody_zones.c


```

    }
    else answer='0';
    switch(answer) {
    case '1' : ad->marked_cancelled = 1; break;
    case '2' : ad->next->marked_cancelled = 1; break;
    default:
        record_err(INTERSPECTING_ALIGNMENTS,
            ad->sim_percent, ad->next->sim_percent);
        align_data_list_free(adl);
        align_data_list_free(czm);
        return NULL;
    }
}

current_should_survive_post =
(pure_seglen(entering_profile, ad->start_est, ad->end_est) >=
HIGH_LEN_BOUND ||
/* t.i.o. don't want quality to */
/* be a factor here?
range_for_gluing(pure_len_intersection_segment));

assert (ad->marked_cancelled == 0 || ad->marked_cancelled == 1);
current_stay = ad->marked_cancelled == 0 &&
(current_should_survive_pre || current_should_survive_post);
if (current_stay) {
    if (last_free!=NULL)
        last_free->end_est = ad->start_est-1;
    *pczm = align_data_new(ad, -1, -1);
    pczm = &((*pczm)->next);
    last_free=NULL;
}
}
else {
    rules_usage_message(0.3, "removing floating alignment", pure_seglen
        (entering_profile, ad->start_est, ad->end_est));
    if (ad->start_est==0) {
        *pczm = align_data_new(NULL, 0, ad->end_est-1);
        last_free=pczm;
        pczm = &((*pczm)->next);
    }
    else
        if (last_free!=NULL)
            last_free->end_est = ad->end_est;
}

next_should_survive = /* does current make next 'want' to stay? */
(pure_seglen(entering_profile, (ad->next)->start_est,
(ad->next)->end_est) >= HIGH_LEN_BOUND ||
current_stay && range_for_gluing(pure_len_intersection_segment));

if (pure_len_intersection_segment <= -2) /* there is something between
the two alignment in
the netering profile */
{
    if (last_free!=NULL)
        last_free->end_est=ad->next->start_est-1;
    else {
        *pczm = align_data_new(NULL, ad->end_est+1, ad->next->start_est-1);
        last_free = *pczm;
        pczm = &((*pczm)->next);
    }
}

/* taking care of last ad in the list (was not in the loop as 'current') */
current_should_survive_post =

```

```

(pure_seglen(entering_profile, ad->start_est, ad->end_est)
>= HIGH_LEN_BOUND);
if (current_should_survive_pre || current_should_survive_post) {
    *pczm = align_data_new(ad, -1, -1);
    last_free = NULL;
    pczm = &((*pczm)->next);
}
else if (last_free!=NULL)
    last_free->end_est = ad->end_est;
/* taking care of (possible) last free segment */
if (len_est - 1 - ad->end_est > 0)
    if (last_free!=NULL)
        last_free->end_est=len_est-1;
else {
    *pczm = align_data_new(NULL, ad->end_est+1, len_est-1);
    pczm = &((*pczm)->next);
}
align_data_list_free(adl);
return czm;
}

/*=====
int merge_node_information (align_data *node_align_list,
                           align_data **graph_align_list)
/* append to graph_align_list the contents of node_align_list in the
following way:
* (i) the final list will not include comparable alignments (one
* containing the other).
* (ii) the list is sorted according to the start points of the alignments
* (in the est).
*/

{
    align_data *p, **q, *r, *ptmp;
    int p_taken_care;
    /* avneramit, dont we want to free node_align_list in the end? */
    for (p = node_align_list; p != NULL; p = p->next) {
        p_taken_care=0;
        for (q = graph_align_list; *q != NULL; q = &((*q)->next)) {
            switch (compare_alignments(p, *q)) {
            case -1 :
                p_taken_care = 1;
                if (seglen(p) >= HIGH_LEN_BOUND) {
                    char answer;
                    intersection_message(p, *q);
                    if (interactives=='y') {
                        printf("cancel the smaller ?");
                        answer = getc(stdin);
                    }
                    else answer='n';
                    if (answer != 'y') {
                        record_err(INTERSPECTING_ALIGNMENTS,
                            p->sim_percent, (*q)->sim_percent);
                        return 0;
                    }
                }
            }
        }
        break;
    case 1:
        if (seglen(*q) >= HIGH_LEN_BOUND) {
            char answer;

```

```

intersection_message(p,*q);
if (interactive=='y') {
    printf("cancel the smaller ?");
    answer = getc(stdin);
}
else answer='n';
if (answer != 'y') {
    record_err(INTERSECTING_ALIGNMENTS,
        p->sim_percent, (*q)->sim_percent);
    return 0;
}
}
free ((*q)->align_str);
align_data__copy(p,*q);
for (r = (*q)->next; r != NULL && r->end_est < p->end_est; r = ptmp) {
    /* Another node needs to be deleted */
    ptmp->r->next;
    align_data_free(r);
}
(*q)->next = r;
p_taken_care = 1;
break;
case 0:
    p_taken_care=0;
    if ((*q)->start_est <= p->start_est) {
        if ((*q)->next==NULL) {
            /* The new item should be inserted right after the current */
            /* one add a new node */
            r = align_data_new(p,-1,-1);
            r->next = (*q)->next;
            (*q)->next = r;
            p_taken_care=1;
        }
        else {
            /* The new item should be inserted right BEFORE the current */
            r = align_data_new(p,-1,-1);
            r->next = *q;
            *q = r;
            p_taken_care=1;
        }
        break;
    }
    if (p_taken_care) break;
}
if (!p_taken_care) {
    /* the current p wasn't thrown away or */
    /* threw other members of the graph's list, */
    /* or was introduced to that list. it */
    /* should now (note *q == NULL) */
    *q = align_data_new(p,-1,-1);
    (*q)->next = NULL;
}
return 1;
}

static int compare_alignments(align_data *first, align_data *second)
{
    if (first->start_est < second->start_est &&
        first->end_est > second->end_est) return 1;

```

```

if (first->start_est > second->start_est &&
    first->end_est < second->end_est) return -1;
return 0;
}

/* ===== */
/* low level functions */
/* ===== */
int seglen(align_data *czm)
{
    return czm->end_est+1-czm->start_est;
}

int pure_seglen(cprof profile, int start, int end)
{
    int loc, count = 0;
    if (profile == NULL) /* in phase one we don't have a profile */
        return end+1-start;
    for (count=0, loc=start; loc <= end; loc++)
        if (cprof_node_char(profile, loc, NULL, NULL) != 'X')
            count++;
    return count;
}

/* Returns the number of chars (including X's) between start and end in the sequ
ence
* corresponding to the profile.
*/
int unpure_seglen(cprof profile, int start, int end)
{
    int loc, count, in_count = 0;
    for (count=0, loc=0; count <= end && loc < profile->len; loc++) {
        if (cprof_node_char(profile, loc, NULL, NULL) != 'X') {
            count++;
            if (count >= start && count <= end)
                in_count++;
        }
        return in_count;
    }

    void czm__shift_left(align_data *czm, int shift)
    {
        czm->start_node--shift;
        czm->end_node--shift;
    }

    void czm__shift_right(align_data *czm, int shift)
    {
        czm->start_node++shift;
        czm->end_node++shift;
    }

    void czm__shift_right_of_same_node(align_data *czm, int id, int shift)
    {
        while (czm)
        {
            if (czm->node_id == id)
                czm__shift_right(czm, shift);
            czm = czm->next;
        }
    }

```

```

    }
}

void align_data___assign(align_data *ad,int node_id,int start_node,int end_node,
int prefix,int suffix)
{
    ad->node_id=node_id;
    ad->start_node = start_node;
    ad->end_node = end_node;
    ad->prefix=prefix;
    ad->suffix = suffix;

    void cut_align_str(align_data *czm,int offset)
    {
        char *tmp_str = (char *)strdup(czm->align_str + offset);
        free(czm->align_str);
        czm->align_str = tmp_str;
    }

    int get_next_zone_boundary(align_data **czm,
                                align_data **ad_west,align_data **ad_east)
    /*
     * get the current and next item in the czm, and make next one the current.
     * assumes the current item is not NULL
     */
    {
        assert (*czm != NULL);
        *ad_west = *czm;
        if ((*czm = (*czm->next)) == NULL)
            return 0;
        *ad_east = *czm;
        return 1;
    }

    void czm___print(align_data *czm)
    {
        char left[5],right[5];
        int first=1,last=0;

        printf("czm = ");
        while (czm)
        {
            if (czm->next==NULL)
                last=1;
            if (czm->node_id == FREE_SEG)
                printf("(free %d..%d) -> ",czm->start_est,czm->end_est);
            else
            {
                left[0]=right[0]=0;
                if ((first) printf(left,"%c)",(czm->prefix)?'+':'-'));
                if ((last) printf(right,"%c)",(czm->suffix)?'+':'-'));
                printf("(node %d, %s%d..%d%s) -> ",czm->node_id,left,czm->start_est,c
                zm->end_est,right);
            }
            czm=czm->next;
            first=0;
        }
    }
}

```

```

    }
    printf("||\n");
}

align_data *align_data_new (align_data *old,int start_est,int end_est)
/*
 * allocating memory for a new ad struct.
 * the contents will those of 'old' if it is not NULL, or otherwise
 * it is opened as a (czm) free segment, with start_est, end_est the
 * boundaries. The first case serves us both for ad and for czm types
 */
{
    align_data *new = (align_data *)malloc(sizeof(align_data));
    if (new==NULL)
        err("align_data_new: memory failure trying to allocate for 'new'");
    if (old==NULL)
    {
        new->node_id = FREE_SEG;
        new->score = 0;
        new->start_est = start_est;
        new->start_node = -1;
        new->end_est = end_est;
        new->end_node = -1;
        new->prefix = 0;
        new->suffix = 0;
        new->marked_cancelled = 0;
        new->id_percent = 0;
        new->sim_percent = 0;
        new->align_str = NULL;
    }
    else
    {
        align_data___copy(old,new);
        new->next=NULL;
        return new;
    }

    static void align_data___copy (align_data *from, align_data *to)
    {
        to->node_id = from->node_id;
        to->score = from->score;
        to->start_node = from->start_node;
        to->start_est = from->start_est;
        to->end_node = from->end_node;
        to->end_est = from->end_est;
        to->prefix = from->prefix;
        to->suffix = from->suffix;
        to->id_percent = from->id_percent;
        to->sim_percent = from->sim_percent;
        to->marked_cancelled = from->marked_cancelled;

        to->align_str = (char *)subseq (from->align_str, 0,
                                        strlen (from->align_str));
        if (to->align_str == NULL)
            err("align_data___copy: memory failure in allocating for 'to->align_str'");
    }

    void align_data___print(align_data *ad)
    {
        printf("node%d <-> new-seq      %d..%d | %d..%d [%d.%d,%d.%d]\n",

```



```

score+=gapext;
if (is_gap_symbol(astr2[astr_idx2-1],2))
    score+=gapop;
node2_loc++;
astr_idx2++;
}
/* there should be some check of the match scores here */

if (score>best_score) {
    best_astr_idx1=astr_idx1;
    best_astr_idx2=astr_idx2;
    best_est_loc = est_loc;
    best_node1_loc = node1_loc;
    best_node2_loc = node2_loc;
}

est_loc++;
if (is_gap_symbol(astr1[astr_idx1],1))
    node1_loc++;
if (is_gap_symbol(astr2[astr_idx2],1))
    node2_loc++;
astr_idx1++;
astr_idx2++;
}

ad1->end_est=best_est_loc;
ad1->end_node=best_node1_loc;
ad1->align_str[best_astr_idx1+1]='\0';
ad2->start_est=best_est_loc+1;
ad2->start_node=best_node2_loc;
tmp = ad2->align_str;
ad2->align_str=strdup(ad2->align_str+best_astr_idx2);
free(tmp);
}

/* take the alignment czm suggests, and make it <offset> shorter at it's start,
 * in terms of the new sequence or of the node according to the parameter
 * <of_node>. update other fields in <czm> (align_str, prefix) accordingly.
 */
czm__shorten_start(align_data *czm,int of_node,int offset)
{
    int offset_node=0,offset_est=0;
    char *old_str=czm->align_str;

    while ((!(of_node)?offset_node:offset_est)<offset)
    {
        if (is_gap_symbol(czm->align_str[0],2))
            offset_est++;
        if (is_gap_symbol(czm->align_str[0],1))
            offset_node++;
        czm->align_str++;
    }
    czm->align_str = strdup(czm->align_str); /* yes, I know it looks strange */
    free(old_str);
    czm->start_node+=offset_node;
    czm->start_est+=offset_est;
    czm->prefix = (czm->start_node < indep_node_len); /* should use pure_seglen? */
}

```

```

static int is_gap_symbol(char c,int which)
{
    switch (which) {
    case 1:
        return (c==ALIGN_GAP1);
        break;
    case 2:
        return (c==ALIGN_GAP2);
        break;
    default:
        err("internal error: is_gap_symbol(..,&d) is not a legal format",which);
    }
}

void update_replace_czm_of_same_node(splice_graph *graph,align_data *czm,
int old,int new,int offset,
int offset_first)
/*
 * (i) replace all occurrences of old with new
 * (ii) update the align data for those occurrences
 * by offset translation. Do the same in the appropriate est zones.
 * if the translations suggest a problem in the consistency of the
 * alignments (does not preserve order), return 0. if small fuzzies,
 * ignore the appropriate 'tail' of the alignment.
 */
{
    int first=1;

    do {
        if (czm->node_id==old) {
            if (new!=DUMMY)
                czm->node_id=new;
            czm->start_node -= (first)?offset_first:offset;
            czm->end_node -= (first)?offset_first:offset;
            assert(czm->start_node > -align_intersec_low);
            if (pure_seglen(graph->node_list[czm->node_id]->profile,
                0,czm->start_node) < indep_node_len)
                czm->prefix=1;
            if (czm->start_node < 0) {
                cut_align_str(czm,-czm->start_node);
                czm__shorten_start(czm,1,-czm->start_node);
                czm->start_node = 0;
            }
            first=0;
        } while (czm=czm->next);
    }

    /*
     * czm->to_node_path -
     * create the list of nodes through which the profile (the one defines the
     * czm) goes through.
     * in case the profile goes through the same node with another node in
     * between, we devide the node according to the czm.
     */
    path_node *czm_to_node_path(align_data *czm,splice_graph *graph)
    {
        align_data *p=czm,*prev=NULL;

```

```

path_node *pn,*last,*node_list=NULL;

while (p) {
    if (p->node_id != -1 && (!prev || prev->node_id != p->node_id)) {
        pn = path_node_new(p->node_id);
        pn->start = get_start_cut_point(p,czm);
        pn->end = get_end_cut_point(p,graph);

        if (node_list == NULL)
            node_list = last = pn;
        else {
            last->next = pn;
            last = pn;
        }

        if (p->node_id != -1)
            prev=p;
        p=p->next;
    }
    return node_list;
}

path_node *path_node_new(int node_id)
{
    path_node *pn;
    if ((pn = (path_node*)malloc(sizeof(path_node))) == NULL)
        err("path_node_new: memory failure allocating %d\n", sizeof(path_node));
    pn->node_id = node_id;
    pn->start = -1;
    pn->end = -1;
    pn->next = NULL;
    return pn;
}

/*
 * get_start_cut_point -
 * return the start point in which we should cut node p->node_id for the
 * path_node according to the czm.
 * Note: we assume that p is it first in its block of ad with the same node_id.
 */
int get_start_cut_point(path_node *p, align_data *czm)
{
    int idx=0;
    align_data *prev_same;
    for (; czm; czm=czm->next) {
        if (czm == p) {
            if (idx==0) {
                return 0;
            }
            else {
                return (p->start_node-1 + prev_same->end_node)/2+1;
            }
        }
        if (czm->node_id == p->node_id) {
            idx++;
            prev_same = czm;
        }
    }
}

```

```

}
err("Error in get_start_cut_point: this point should not be reached\n");
}
/*
 * get_end_cut_point -
 * return the end point in which we should cut node p->node_id for the
 * path_node according to the czm.
 */
int get_end_cut_point(align_data *p, splice_graph *graph)
{
    int id=p->node_id;
    align_data *ad;

    /* Go to the end of p's block */
    while (p->next != NULL && id == next_real_node(p)) p=p->next;

    for (ad=p->next; ad; ad=ad->next) {
        if (ad->node_id == p->node_id)
            return (ad->start_node-1 + p->end_node)/2;
    }
    return splice_node__len(graph->node_list[p->node_id]) - 1;
}

int next_real_node(align_data *p)
{
    p=p->next;
    while (p && p->node_id == -1)
        p=p->next;
    if (p) return p->node_id;
    else return -1;
}

/*
 * should_cut_node -
 * We should cut a node if the czm looks some thing like this:
 * (node i)->...->(node j)->...->(node i)
 */
int should_cut_node(align_data *czm, int node_id)
{
    int started=0, stopped=0;
    for (; czm; czm=czm->next) {
        if (!started && node_id == czm->node_id)
            started=1;
        if (started && node_id != czm->node_id)
            stopped=1;
        if (stopped && node_id == czm->node_id)
            return 1;
    }
    return 0;
}

path_node *revers_list(path_node *list)
{
    path_node *pn;
    if (list == NULL || list->next == NULL) return list;
    pn=list;
    list=revers_list(pn->next);
    pn->next->next=pn;
    pn->next=NULL;
}

```

```

return list;
}

/* new_adl_to_czm -
 * the algorithm is as follows:
 * 1. we call the old adl_to_czm (this should be changed since there are some
 *    unnecessary action)
 * 2. call czm_to_node_path which retrun a list of path_node with the
 *    information on which nodes we should unify to create a path containing
 *    the entering profile.
 * 3. call node_path_to_cprof and create the path profile.
 * 4. find the alignmet of the entering profile to the path profile.
 * 5. call break_path_adl which creates align_data list for the nodes of the
 *    path.
 * 6. call the old adl_to_czm.
 */
align_data *new_adl_to_czm(align_data *adl, int len_est, cprof entering_profile,
                           splice_graph *graph)
{
    cprof_align cp_align_list;
    cprof path_cprof;
    align_data *path_adl, *czm, *new_czm;
    path_node *node_path;

    czm = adl_to_czm(adl, len_est, entering_profile);
    if (czm == NULL) {
        return NULL;
    }
    printf("czm after adl_to_czm: \n");
    align_data_list__print(czm);

    node_path = czm_to_node_path(czm, graph);
    if (node_path == NULL)
        return czm;
    path_node__print(node_path);

    path_cprof = node_path_to_cprof(node_path, graph);
    /* cprof_dump(path_cprof); */

    cp_align_list = cprof_alignment(path_cprof, entering_profile,
                                     SEPARATE_HILLTOPS|local_overlap_mode);
    path_adl = cprof_align_to_align_data(cp_align_list, path_cprof,
                                         entering_profile, -1);
    cprof_align_free(cp_align_list);

    printf("path_adl: \n");
    align_data_list__print(path_adl);

    break_path_adl(spath_adl, path_cprof, node_path, graph);
    cprof_free(path_cprof);
    path_node_list__free(node_path);

    printf("path_adl after break_path_adl: \n");
    align_data_list__print(path_adl);

    if (path_adl == NULL)
        return czm;
    new_czm = adl_to_czm(path_adl, len_est, entering_profile);
    printf("new_czm: \n");

```

```

align_data_list__print(new_czm);

if (!czm_is_same(czm, new_czm)) {
    printf("new_czm is different from hte original\nnew_czm: \n");
    czm__print(new_czm);
    printf("czm: \n");
    czm__print(czm);
}
return new_czm;
}

int czm__is_same(align_data *czm1, align_data *czm2)
{
    while(czm1 && czm2) {
        if (czm1->node_id != czm2->node_id ||
            czm1->start_est != czm2->start_est ||
            czm1->end_est != czm2->end_est ||
            czm1->end_node != czm2->end_node)
            return 0;
        czm1 = czm1->next;
        czm2 = czm2->next;
    }
    if (czm1 != NULL || czm2 != NULL)
        return 0;
    return 1;
}

cprof node_path_to_cprof(path_node *node_path, splice_graph *graph)
{
    path_node *pn;
    cprof path_prof, p;

    if (node_path == NULL) return NULL;

    path_prof = cprof_segment(graph->node_list[node_path->node_id]->profile,
                              node_path->start, node_path->end);
    for (pn = node_path->next; pn; pn = pn->next) {
        p = cprof_segment(graph->node_list[pn->node_id]->profile,
                          pn->start, pn->end);
        cprof_glue(path_prof, p, e_after, 1 /*should be changed !!!*/);
    }
    return path_prof;
}

void break_path_adl(align_data **p_adl, cprof path_cprof,
                   path_node *node_path, splice_graph *graph)
{
    align_data *ad, *path_adl = *p_adl;
    int start_node = 0;
    int end_node = path_node__len(node_path);

    while(path_adl) {
        if (pure_seglen(path_cprof, path_adl->start_node, path_adl->end_node) <
            indep_node_len) {
            printf("break_path_adl removed align_data: \n");
            align_data__print(path_adl);
            ad = path_adl->next;
            align_data_list__remove(p_adl, path_adl);
        }
    }

```



```

align_data_free(path_adl);
path_adl = ad;
continue;
}
if (path_adl->start_node < end_node && path_adl->end_node >= end_node) {
    align_data_split(path_adl, end_node, path_cprof, 1);
    continue;
}
/* remove this code */
path_adl->node_id = node_path->node_id;
path_adl->start_node = start_node - node_path->start;
path_adl->end_node = start_node - node_path->start;
path_adl->prefix =
    (pure_seglen(graph->node_list[path_adl->node_id]->profile,
0, path_adl->start_node - 1) < indep_node_len);
path_adl->suffix = 1;

path_adl = path_adl->next;
node_path = node_path->next;
start_node = end_node;
end_node += path_node__len(node_path);
}
else if (path_adl->start_node < end_node &&
    path_adl->end_node < end_node) {
    path_adl->node_id = node_path->node_id;
    path_adl->start_node = start_node - node_path->start;
    path_adl->end_node = start_node - node_path->start;
    path_adl->prefix =
        (pure_seglen(graph->node_list[path_adl->node_id]->profile,
0, path_adl->start_node - 1) < indep_node_len);
    path_adl->suffix =
        (pure_seglen(graph->node_list[path_adl->node_id]->profile,
path_adl->end_node + 1,
graph->node_list[path_adl->node_id]->profile->len-1)
< indep_node_len);
    path_adl = path_adl->next;
}
else if (path_adl->start_node >= end_node &&
    path_adl->end_node >= end_node) {
    node_path = node_path->next;
    start_node = end_node;
    end_node += path_node__len(node_path);
}
else
    err("Internal error in break_path_adl\n");
}
}
/*
 * align_data_split -
 * split AD at a given CUTPOINT of P (P is the node if WHICH is 1 else the
 * est). the score and percentage fields are NOT updated.
 * CUTPOINT will be in the right align_data after the split.
 * fuzziness:
 * 1. cutpoint is not well defined if there are gaps in P before the
 * cutpoint, in this case we simply put all the gaps in the left side

```

```

* 2. if after the split we are left with too small alignmmet we throu it.
*/
void align_data_split(align_data *ad, int cutpoint,
    cprof pl, int which)
{
    align_data *right;
    int idx, i1, i2;
    for (idx=0, i1=ad->start_node, i2=ad->start_est;
        ((which=1)?i1:i2) < cutpoint;
        idx++) {
        if (!is_gap_symbol(ad->align_str[idx], 2))
            i2++;
        if (!is_gap_symbol(ad->align_str[idx], 1))
            i1++;
    }
    right = align_data_new(NULL, i2, ad->end_est);
    right->node_id = ad->node_id;
    right->start_node = i1;
    right->start_est = i2;
    right->end_node = ad->end_node;
    right->end_est = ad->end_est;
    right->align_str = strdup(ad->align_str+idx);
    right->next = ad->next;
    ad->end_est = i2-1;
    ad->end_node = i1-1;
    ad->align_str[idx]='0';
    ad->next = right;
}

int path_node__len(path_node *pn)
{
    return pn->end+1 - pn->start;
}

void path_node__print(path_node *pn)
{
    while(pn) {
        printf("(node%d %d. %d)", pn->node_id, pn->start, pn->end);
        if (pn->next)
            printf("->");
        pn=pn->next;
    }
    printf("\n");
}

void path_node_list__free(path_node *pn)
{
    path_node *p;
    while(pn) {
        p=pn->next;
        path_node__free(pn);
        pn=p;
    }
}

void path_node__free(path_node *pn)
{
    free(pn);
}

```



```

/*
 * Copyright 1996 Compugen, Ltd.
 * Authorization to use this code is given solely to Compugen customers.
 * This authorization is limited by the terms of the Compugen Hardware and
 * Software License Agreement.
 *
 * Last update: $Date: 1998/02/22 17:20:07 $ by $Author: ariels $
 * Revision: $Revision: 1.7 $
 */

```

```

/*
 * Hacked by ariels@vivaldi 6 October 1997 to return also
 * non-overlapping alignments, depending on an option.
 */

```

```

/*
 * $Log: dp.c,v $
 * Revision 1.7 1998/02/22 17:20:07 ariels
 * Change various 'harmless' printf()'s to fatal err()'s
 *
 * Revision 1.6 1998/01/17 11:23:42 eyal
 * Fix bug in calc_id_sim.
 *
 * Revision 1.5 1998/01/15 20:25:00 eyal
 * Fix bug in calc_id_sim() [D... [D
 *
 * Revision 1.4 1998/01/14 15:24:09 eyal
 * Adding function calc_id_sim
 *
 * Revision 1.3 1997/12/01 22:20:54 avner
 * Fixed a bug in 'improve_six18_result', that caused the diff structure to be
 * changed with no use. This should fix the wrong figures we got for est-error
 * when the splice-graph was nontrivial.
 *
 * Revision 1.2 1997/11/30 07:52:37 ariels
 * Added ChangeLog comment and static rcsid string.
 */

```

```
static char rcsid[] = "$Id: dp.c,v 1.7 1998/02/22 17:20:07 ariels Exp $";
```

```

#include <stdio.h>
#include <strings.h>
#include <sys/types.h>
#include <malloc.h>
#include <math.h>
#include "dp.h"

```

```

/* Macros */
#define MAT(i,j) (mat + ((i)*leny + (j)))

/* definition of constants */
#define INF 100000

#define NONE -1
#define DIAG 0
#define LEFT 1

```

dp.c

```

#define DOWN 2
#define XGAP 3
#define YGAP 4
#define ALT 5

/* Typedefs */
typedef struct node {
    double match, ins, del;
} node;

typedef struct lnode {
    double sc[3];
    int x0[3], y0[3], s0[3];
} lnode;

typedef struct sixnode {
    double match, ins, del, xgap, ygap, alt;
} sixnode;

typedef struct lsixnode {
    double sc[6];
    int x0[6], y0[6], s0[6];
} lsixnode;

/* Static variables */
int X_GO, X_GE, Y_GO, Y_GE;
int sc[26][26]; /* ALPHABET is a subset of the English Alphabet */
double exact_sc[26][26];

void dbg_print(char x[], char y[], node *mat);
void get_match(char x[], char y[], node *mat, int it, int jt, hilltop *ht);
void get_six18_match(char x[], char y[], sixnode *mat, int it, int jt, int st,
                    hilltop *ht, int mode);
void call_linear_six18_alignment(char x[], char y[], hilltop *ht, int mode);
void linear_six18_alignment(char x[], char y[], hilltop *ht, int bs, int es,
                           int mode);
void improve_six18_result(hilltop *ht, int mode);
void get_band_match(char x[], char y[], int x0, int y0, int width, node *mat,
                    int it, int jt, hilltop *ht);

/*=====
void init_matrix(int match, int mismatch, int tGE, int tGO)
/*
 * Initialize static variables for nucleotides. We use the following
 * nomenclature:
 * K = G or T, M = A or C, R = G or A, S = G or C, W = A or T, Y = T or C
 * B = G or T or C, D = G or A or T, H = A or C or T, V = G or C or A
 * N = A or C or G or T, X = None of the above.
 */
{
    int i, j, len, good, bad;
    char alphabet[] = "ACMGKRSVWYWKHKN";
    int pop[16] = {0, 1, 1, 2, 1, 2, 2, 3, 1, 2, 2, 3, 2, 3, 3, 4};
    double maxc, minc, val, c, p, q, exp(), log();
    /*
     * Approximate the multiplicative constant c used to compute the
     * match and mismatch user-controlled constants
     */
}

```

dp.c

```

if (match <= 0 || mismatch >= 0 || match > (-3*mismatch))
    error(1, -1, "Bad values for match (%d) and mismatch (%d)",
        match, mismatch);
maxc = 100000.0;
minc = 0.1;
for (i = 0; i < 100; i++) {
    c = (maxc + minc) / 2;
    val = (exp((double)match/c) + 3*exp((double)mismatch/c))/4;
    if (val < 1.0) maxc = c;
    else minc = c;
}
p = exp(match/c)/16;
q = exp(mismatch/c)/16;
/* printf ("p = %g, q = %g, c = %g\n", p, q, c); */
for (i = 0; i < 26; i++)
    for (j = 0; j < 26; j++)
        sc[i][j] = exact_sc[i][j] = mismatch;
/*
 * For each letter of the alphabet, find the correct score by
 * aposteriori calculation, using p, q, and c.
 */
for (i = 1; i < 16; i++)
    for (j = 1; j < 16; j++) {
        good = pop[i] * pop[j] - good;
        bad = pop[i] * pop[j] - good;
        val = c * log(16 * (good * p + bad * q) / (good + bad));
        exact_sc[alphabet[i] - 'A'][alphabet[j] - 'A'] = val;
        sc[alphabet[i] - 'A'][alphabet[j] - 'A'] = (int) (val + 0.5 * (val > 0));
    }
}

/*
 * for (i = 0; i < 26; i++) {
 *     for (j = 0; j < 26; j++) {
 *         if (exact_sc[i][j] >= 0) printf (" ");
 *         printf ("%2.2f", exact_sc[i][j] / 100);
 *     }
 *     printf ("\n");
 * }
 * fflush (stdout); */
X_GO = tGO;
X_GE = tGE;
Y_GO = tGO;
Y_GE = tGE;
}

/*=====
void get_alignment (char in_x[], char in_y[], hilltop *ht, int where)
/*
 * Dynamic programming routine to match two nucleotide sequences, using
 * square memory.
 * From the contents of "ht" we figure out whether to compute the whole matrix
 * or to cut above a certain diagonal:
 *
 * -----
 * |*****| the given value of loop_end is the
 * |*****| height of the vertical bar in the drawing.
 * |-----|
 *
 * The "where" parameter can be either BEST (find best score) or CORNER (force

```

```

* the routine to take the upper right corner).
*/
int i, j, best_i = 0, best_j = 0, tmp, id_count, sim_count, no_n_count;
int lenx, leny, mode, loop_end;
double max_score = 0.0;
node *diag, *left, *down, *current, *mat, zero;
hilltop *res;
char *x, *y;

/* copy input strings to working strings, shorten if necessary */
x = subseq (in_x, ht->x0, ht->xt); lenx = strlen (x);
y = subseq (in_y, ht->y0, ht->yt); leny = strlen (y);

if (lenx * leny > MAX_FULL * MAX_FULL) {
    /* We need a linear memory alignment here (which is slower) */
    linear_alignment (in_x, in_y, ht, 0, 0);
    free (x);
    free (y);
    ht->final = 1;
    ht->len = strlen (ht->x);
    id_count = sim_count = no_n_count = 0;
    for (i = 0; i < strlen (ht->diff); i++) {
        id_count += (ht->diff[i] == '|');
        if (ht->x[i] != '-' && ht->y[i] != '-')
            sim_count += (exact_sc[ht->x[i] - 'A'][ht->y[i] - 'A'] > 0);
        if (ht->x[i] != 'N' && ht->y[i] != 'N') no_n_count++;
    }
    ht->id_percent = 100 * (double) (id_count) / (double) (no_n_count);
    ht->sim_percent = 100 * (double) (sim_count) / (double) (no_n_count);
    return;
}

if (ht->type == TANDEM) {
    loop_end = ht->x0 - ht->y0;
    if (loop_end > leny) loop_end = leny;
} else {
    loop_end = leny;
}

/* allocate memory and initialize matrix boundaries */
if (!mat = (node *) malloc (1 * lenx * leny * sizeof (node)))
    error(1, -1, "couldn't allocate memory for nodes 1 (%d %d)\n",
        lenx, leny);

zero.mat = zero.ins = zero.del = 0;
for (j = 0; j < lenx; j++) {
    if (j + loop_end >= leny) break;
    if (j + loop_end < 0) continue;
    current = &mat[j * leny + j + loop_end];
    *current = zero;
}

/* main loop */
for (i = 0; i < lenx; i++) {
    /* Advance one column and initialize pointers */
    current = &mat[i * leny];
    if (i == 0) left = &zero;
    else left = &mat[(i - 1) * leny];
    diag = &zero; down = &zero;

```

```

for (j = 0; j < loop_end; j++) {
    current->match = diag->match;
    if (diag->ins > current->match) current->match = diag->ins;
    if (diag->del > current->match) current->match = diag->del;
    current->match += exact_sc[x[i]-'A'][y[j]-'A'];
    if (current->match < 0) current->match = 0;

    current->ins = left->match - X_GO;
    if (left->ins - X_GE > current->ins) current->ins = left->ins - X_GE;
    if (current->ins < 0) current->ins = 0;

    current->del = down->match - Y_GO;
    if (down->del - Y_GE > current->del) current->del = down->del - Y_GE;
    if (current->del < 0) current->del = 0;

    /* Save best node */
    if ((current->match) >= max_score) {
        max_score = current->match; best_i = i; best_j = j;
    }

    /* Advance pointers for next node */
    if (i > 0) (diag = left; left++;)
    down = current; current++;
    if (loop_end < leny) loop_end++;
}

if (where == CORNER) {
    best_i = lenx-1;
    best_j = leny-1;
}

/*for (j = leny - 1; j >= 0; j--) {
    for (i = 0; i < lenx; i++)
        printf (" %2d ", mat[i*leny+j].match);
    printf ("\n");
    for (i = 0; i < lenx; i++)
        printf ("%2d %2d ", mat[i*leny+j].ins, mat[i*leny+j].del);
    printf ("\n\n");
}

get_match (x, y, mat, best_i, best_j, ht);
free (mat);
free (x);
free (y);
}

/*=====
void get_match (char x[], char y[], node *mat, int it, int jt, hilltop *ht)
/* Extract the best solution */
{
    char *nice_x, *nice_y, *nice_diff;
    int last, i, j, loc, end, id_count, sim_count, no_n_count, lenx, leny;
    double tmp;
    node *current, *diag, *left, *down;

    lenx = strlen (x);
    leny = strlen (y);

    end = it + jt;
    if (((nice_x = (char *)malloc (end+2)) == NULL) ||
        ((nice_y = (char *)malloc (end+2)) == NULL) ||
        ((nice_diff = (char *)malloc (end+2)) == NULL))

```

```

error (1, -1, "Can't allocate memory for nice_strings");

nice_x[end+1] = nice_y[end+1] = nice_diff[end+1] = '\0';
i = it; j = jt;
last = DIAG;

for (loc = end; last != NONE; loc--) {
    current = &mat[i * leny + j];
    left = &mat[(i-1)*leny + j];
    down = &mat[i * leny + j-1];
    diag = &mat[(i-1)*leny + j-1];

    if (last == DIAG) {
        nice_x[loc] = x[i];
        nice_y[loc] = y[j];
        if (x[i] == y[j] && exact_sc[x[i]-'A'][y[j]-'A'] > 0)
            nice_diff[loc] = '|';
        else nice_diff[loc] = (exact_sc[x[i]-'A'][y[j]-'A'] > 0) ? ':' : ' ';
        tmp = current->match - exact_sc[x[i]-'A'][y[j]-'A'];
        if (tmp == 0) last = NONE;
        else {
            if (fabs(diag->match - tmp) < EPSILON) last = DIAG;
            else if (fabs(diag->del - tmp) < EPSILON) last = DOWN;
            else if (fabs(diag->ins - tmp) < EPSILON) last = LEFT;
            else err("error in back-tracking 1, %d %d\n", i, j);
        }
        i--; j--;
    }
    else if (last == LEFT) {
        nice_x[loc] = x[i];
        nice_y[loc] = '-';
        nice_diff[loc] = ' ';
        if (current->ins == 0) last = NONE;
        else {
            tmp = current->ins;
            if (fabs(left->match - tmp-X_GO) < EPSILON) last = DIAG;
            else if (fabs(left->ins - tmp-X_GO) < EPSILON) last = LEFT;
            else err("error in back-tracking 2, %d %d\n", i, j);
        }
        i--;
    }
    else if (last == DOWN) {
        nice_x[loc] = '-';
        nice_y[loc] = y[j];
        nice_diff[loc] = ' ';
        if (current->del == 0) last = NONE;
        else {
            tmp = current->del;
            if (fabs(down->del - tmp-Y_GO) < EPSILON) last = DOWN;
            else if (fabs(down->match - tmp-Y_GO) < EPSILON) last = DIAG;
            else err("error in back-tracking 3, %d %d\n", i, j);
        }
        j--;
    }
}

ht->final = 1;
ht->len = strlen (nice_y[loc]);
ht->x = subseq (nice_x[loc], 0, ht->len);
ht->y = subseq (nice_y[loc], 0, ht->len);
ht->diff = subseq (nice_diff[loc], 0, ht->len);
free (nice_x);
free (nice_y);
free (nice_diff);

```

```

id_count = sim_count = no_n_count = 0;
for (i = 0; i < strlen(ht->diff); i++) {
    id_count += (ht->diff[i] == '|');
    if (ht->x[i] != '-' && ht->y[i] != '-')
        sim_count += (exact_sc(ht->x[i]-'A')[ht->y[i]-'A'] > 0);
    if (ht->x[i] != 'N' && ht->y[i] != 'N') no_n_count++;
}
ht->id_percent = 100*(double)(id_count) / (double)(no_n_count);
ht->sim_percent = 100*(double)(sim_count) / (double)(no_n_count);
}

/*=====
void dbg_print (char x[], char y[], node *mat)
/* Debug printong of the matrix */
{
    int i, j, lenx, leny;

    lenx = strlen(x);
    leny = strlen(y);
    for (j = leny-1; j >= 0; j--) {
        printf ("%c ", y[j]);
        for (i = 0; i < lenx; i++) {
            printf ("%2d %2d %2d ", MAT(i,j)->match,
                MAT(i,j)->ins, MAT(i,j)->del);
        }
        printf ("\n");
    }
    for (i = 0; i < lenx; i++)
        printf (" %c ", x[i]);
    printf ("\n");
}

/*=====
hilltop *hilltops (char in_x[], char in_y[], int mode, int cutoff, int bound)
/*
* Dynamic programming routine to match two nucleotide sequences, and
* get ALL distinct peaks, using the Hilltops generalization of SW.
* Mode = > 0 : full Smith-Watterman matrix calculation.
*         = 0 : lower left half of matrix only (useful for inverted
*         and tandem repeat searches)
*         = INVERTED : lower left half of matrix only (useful for straight
*         and tandem repeat searches)
* STRAIGHT : lower right half of matrix only (useful for straight
* and tandem repeat searches)
* Scores under bound will not be printed. between two scores above bound
* only the better one will be printed, unless they are separated by a valley
* with scores under cutoff.
* The routine returns the number of alignments found. res will point to a
* linked list of alignments of that length.
*/
{
    *segg, *segg_tmp;
    int i, j, best_i, best_j, best, tmp, segment_flag;
    int lenx, leny, loop_end, loop_add;
    *diag, *left, *down, *current, *col, *prevcol, *tmp_node;
    mnode zero, zero2;
    char *x, *y;
    hilltop *work, *list, **http, *http;

    /* copy input strings to working strings, shorten if necessary */
    x = subseq (in_x, 0, MAX_LEN); lenx = strlen (x);

```

dp.c

```

y = subseq (in_y, 0, MAX_LEN); leny = strlen (y);
list = work = NULL;

if (mode == STRAIGHT || mode == INVERTED) && lenx != leny)
    printf ("Warning: Non-square matrix in triangular mode\n");

/* allocate memory and initialize matrix vertical edge */
if ((col = (mnode *) malloc ((1+leny) * sizeof (struct mnode))) ||
    (prevcol = (mnode *) malloc ((1+leny) * sizeof (struct mnode))))
    error(1, -1, "couldn't allocate memory for nodes 2 (%d).\n", leny);

/* Take care of triangular modes */
loop_end = leny; loop_add = 0;
if (mode == INVERTED) loop_add = -1;
else if (mode == STRAIGHT) { loop_end = 0; loop_add = 1; }

zero.match.s = zero.ins.s = zero.del.s = 0;
zero.match.xy = zero.ins.xy = zero.del.xy = 0;
for (j = 0; j < leny; j++) {
    col[j] = prevcol[j] = zero;
    col[j].match.xy = col[j].ins.xy = col[j].del.xy = (j+1) & 0xffff;
    /* 16 lsbits are for y0, 16 msbits for x0. This is enough assuming
       no ALIGNMENT is longer than 65535 nucleotides */
}

/* main loop */
for (i = 0; i < lenx; i++) {
    /* Advance one column and initialize pointers */
    tmp_node = col; col = prevcol; prevcol = tmp_node;
    left = prevcol; current = col;
    diag = &zero2; down = &zero;

    segment_flag = 0; segg = NULL;

    /* Initialize horizontal edge */
    *diag = *down;
    down->match.xy = down->ins.xy = down->del.xy = (((i+1) & 0xffff) << 16);

    /* Loop over nodes in the column */
    for (j = 0; j < loop_end; j++) {
        current->match = diag->match;
        if (diag->ins.s > current->match.s) current->match = diag->ins;
        if (diag->del.s > current->match.s) current->match = diag->del;
        current->match.s += exact_sc(x[i]-'A')[y[j]-'A'];
        if (current->match.s <= 0) {
            current->match.s = 0;
            current->match.xy = (((i+1) & 0xffff) << 16) ^ ((j+1) & 0xffff);
        }

        current->ins = left->match;
        current->ins.s -= X_GO;
        if (left->ins.s - X_GE > current->ins.s) {
            current->ins = left->ins; current->ins.s -= X_GE;
        }
        if (current->ins.s <= 0) {
            current->ins.s = 0;
            current->ins.xy = (((i+1) & 0xffff) << 16) ^ ((j+1) & 0xffff);
        }

        current->del = down->match;

```

dp.c


```

current->del.s -= Y_GO;
if (down->del.s - Y_GE > current->del.s) {
    current->del = down->del; current->del.s -= Y_GE;
}
if (current->del.s <= 0) {
    current->del.s = 0;
    current->del.xy = ((i+1) & 0xffff) << 16 | ((j+1) & 0xffff);
}

/* Save transitions of cutoff */
if (!segment_flag) {
    if ((current->match.s) >= cutoff) {
        create_segment(&seg_tmp);
        seg_tmp->next = segp;
        segp->start = segp->best_y = j; segp->end = loop_end-1;
        segp->score = current->match.s; segp->best_y = j;
        segp->x0 = current->match.xy;
        segment_flag = 1;
    }
} else {
    if (current->match.s > segp->score) {
        segp->score = current->match.s; segp->best_y = j;
        segp->x0 = current->match.xy;
    }
    if ((current->match.s) < cutoff) {
        segp->end = j-1; segment_flag = 0;
    }
}

/* Advance pointers for next node */
diag = left; left++; down = current; current++;

/* Deal with segments */
update_all_hilltops (segp, &work, i, 0);

/* Copy boundaryless hilltops aside */
for (htpp = &work; htp != NULL; ) {
    if ((htpp->boundary == NULL) ||
        htp == (*htpp);
        htp->type = mode;
        if ((htp->type == STRAIGHT) && (htp->x0 - htp->y0 < TANDEM_GAP))
            htp->type = TANDEM;
        (*htpp) = ((*htpp)->next);
        if (htp->score >= bound)
            trim_hilltop (in_x, in_y, htp, cutoff);
        if (htp->score >= bound) { /* Score may have dropped - check again. */
            htp->next = list;
            list = htp;
        } else destroy_hilltop (&htp);
        htp = &((*htpp)->next);
    }
    loop_end += loop_add;
}

/* At the end there might be some hilltops which still have boundaries */
for (htpp = &work; htp != NULL; ) {
    (*htpp) = ((*htpp)->next);
    htp->type = mode;
    if ((htp->type == STRAIGHT) && (htp->x0 - htp->y0 < TANDEM_GAP))

```

```

htp->type = TANDEM;
if (htp->score >= bound)
    trim_hilltop (in_x, in_y, htp, cutoff);
if (htp->score >= bound) { /* Score may have dropped - check again. */
    destroy_segment_list (&(htp->boundary));
    htp->next = list;
    list = htp;
} else destroy_hilltop (&htp);
}

free (col);
free (prevcol);
free (x);
free (y);
return (list);
}

/*=====
void global_alignment (char in_x[], char in_y[], int loop_end, hilltop **ht)
/*
 * Dynamic programming routine to match two nucleotide sequences, and
 * get the end of the best alignment starting at the beginning of them both.
 * Returns a Hilltop structure.
 * Runs in linear memory.
 * Loop_end is used to cut a piece of the matrix, just like in "get_align".
 */
{
    segment *seggp, **segtmp;
    int i, j, best_i, best_j, tmp, segment_flag;
    double max_score = 0.0;
    int lenx, leny;
    node *diag, *left, *down, *current, *col, *prevcol, *tmp_node, zero, bad;
    char *x, *y;

    /* copy input strings to working strings, shorten if necessary */
    x = subseq (in_x, 0, MAX_LEN); lenx = strlen (x);
    y = subseq (in_y, 0, MAX_LEN); leny = strlen (y);
    if (loop_end > leny) loop_end = leny;
    create_hilltop (ht);

    /* allocate memory and initialize matrix vertical edge */
    if (!col = (node *) malloc (1+leny * sizeof (struct node))) ||
        !prevcol = (node *) malloc (1+leny * sizeof (struct node))) ||
        error(1, -1, "couldn't allocate memory for nodes 3 (%d)\n", leny);
    zero.match = zero.ins = zero.del = 0;
    bad.match = bad.ins = bad.del = -1000000;

    for (j = 0; j < leny; j++) col[j] = prevcol[j] = bad;

    /* main loop */
    for (i = 0; i < lenx; i++) {
        /* Advance one column and initialize pointers */
        tmp_node = col; col = prevcol; prevcol = tmp_node;
        diag = &bad; left = prevcol;
        down = &bad; current = col;
        if (i == 0) diag = &zero;

        /* Loop over nodes in the column */
        for (j = 0; j < loop_end; j++) {

```

```

current->match = diag->match;
if (diag->ins > current->match) current->match = diag->ins;
if (diag->del > current->match) current->match = diag->del;
current->match += exact_sc[x[i]-'A'][y[j]-'A'];
if (current->match <= 0) current->match = bad_match;

current->ins = left->match - X_GO;
if (left->ins - X_GE > current->ins) current->ins = left->ins - X_GE;
if (current->ins <= 0) current->ins = bad_ins;

current->del = down->match - Y_GO;
if (down->del - Y_GE > current->del) current->del = down->del - Y_GE;
if (current->del <= 0) current->del = bad_del;

/* Save maximal value */
if (current->match > (*ht->score) {
    (*ht)->score = current->match;
    (*ht)->xt = i;
    (*ht)->yt = j;
}

/* Advance pointers for next node */
diag = left; left++;
down = current; current++;
}
if (loop_end < leny) loop_end++;
}
free (col);
free (prevcol);
free (x);
free (y);
}

/*=====*/
double smith_waterman (char in_x[], char in_y[])
/*
 * Dynamic programming routine to match two nucleotide sequences, and
 * get the end of the best local alignment.
 * Returns the maximal score.
 * Runs in linear memory.
 */
{
    int i, j, best_i, best_j, best;
    int lenx, leny;
    double max_score = 0.0, *help, *p;
    register double tmp;
    node *diag, *left, *down, *current, *col, *prevcol, *tmp_node, zero;
    char *x, *y;

    /* copy input strings to working strings, shorten if necessary */
    x = subseq (in_x, 0, MAX_LEN);
    y = subseq (in_y, 0, MAX_LEN);
    lenx = strlen (x);
    leny = strlen (y);

    /* allocate memory */
    if (! (col = (node *) malloc (1+leny * sizeof (struct node))) ||
        ! (prevcol = (node *) malloc (1+leny * sizeof (struct node))) ||
        error (1, -1, "couldn't allocate memory for nodes 4 (%d).\n", leny);
        if (! (help = (double *) malloc (1 + leny*26 * sizeof (double))))

```

dp.c

```

error (1, -1, "couldn't allocate memory for help (%d).\n", leny);
for (i = 0; i < 26; i++)
    for (j = 0; j < leny; j++)
        help[i*leny+j] = exact_sc[i][y[j]-'A'];

/* initialize vertical edge */
zero_match = zero_ins = zero_del = 0;
for (j = 0; j < leny; j++) col[j] = zero;

/* main loop */
for (i = 0; i < lenx; i++) {
    /* Advance one column and initialize pointers */
    tmp_node = col; col = prevcol; prevcol = tmp_node;
    diag = &zero; left = prevcol;
    down = &zero; current = col;
    p = &help[(x[i] - 'A')*leny];

    /* Loop over nodes in the column */
    for (j = 0; j < leny; j++) {
        current->match = diag->match;
        if (diag->ins > current->match) current->match = diag->ins;
        if (diag->del > current->match) current->match = diag->del;
        /* current->match += exact_sc[x[i]-'A'][y[j]-'A']; */
        current->match += *(p++);
        if (current->match <= 0) current->match = 0;

        current->ins = left->match - X_GO;
        if (left->ins - X_GE > current->ins) current->ins = left->ins - X_GE;
        if (current->ins <= 0) current->ins = 0;

        current->del = down->match - Y_GO;
        if (down->del - Y_GE > current->del) current->del = down->del - Y_GE;
        if (current->del <= 0) current->del = 0;

        /* Save maximal value */
        if (current->match > max_score)
            max_score = current->match;
        /* Advance pointers for next node */
        diag = left; left++;
        down = current; current++;
    }
}
free (col);
free (prevcol);
free (x);
free (y);
free (help);
return (max_score);
}

/*=====*/
void trim_hilltop (char in_x[], char in_y[], hilltop *ht, int cutoff)
/*
 * Trim an alignment and leave just the end if the score (going back) drops
 * below 0 at any point.
 */
{
    hilltop *p;
    char *tmpx, *tmpy;
    int loop_end;

```

dp.c

```

tmpx = subseq (in_x, ht->xt, ht->x0);
tmpy = subseq (in_y, ht->yt, ht->y0);
loop_end = strlen (tmpx);
if (ht->type == TANDEM || ht->type == STRAIGHT)
    global_alignment (tmpy, tmpx, loop_end, &p);
if (p->score != ht->score) {
    ht->score = ht->score;
    ht->x0 = ht->xt - p->yt;
    ht->y0 = ht->yt - p->xt;
    ht->score = p->score;
}

free (tmpx);
free (tmpy);
}

/*=====
void linear_alignment (char x[], char y[], hilltop *ht, int bs, int es)
/*
* Dynamic programming routine to match two nucleotide sequences, using
* linear memory.
* From the contents of "ht" we figure out whether to compute the whole matrix
* or to cut above a certain diagonal:
* -----
* |***** the given value of loop_end is the
* |***** height of the vertical bar in the drawing.
* |*****
* |*****
* bs and es are the beginning and end states. When the routine is run from
* the outside they should both be 0 (i.e. match state). Upon recursive calls
* either (or both) can also be 1 (insert) or 2 (delete).
* */

int i, j, k, x0, y0, s0, update;
int lenx, leny, loop_end, h_line, v_line;
lnode *diag, *left, *down, *current, *tmp_node;
lnode *col, *prevcol, *zero, *bad;
hilltop *bottom, *top;

/*printf ("In linear_alignment x0=%d, xt=%d, y0=%d, yt=%d, bs=%d, es=%d\n",
    ht->x0, ht->xt, ht->y0, ht->yt, bs, es); */

lenx = ht->xt - ht->x0 + 1;
leny = ht->yt - ht->y0 + 1;
/* First check if this is the end of the recursion */
if (lenx <= 2 && leny <= 2) {
    if (! (ht->x = (char *) malloc (3)))
        error (1, -1, "couldn't allocate memory for alignment string.\n");
    if (! (ht->y = (char *) malloc (3)))
        error (1, -1, "couldn't allocate memory for alignment string.\n");
    if (! (ht->diff = (char *) malloc (3)))
        error (1, -1, "couldn't allocate memory for alignment string.\n");
    ht->x[2] = ht->y[2] = ht->diff[2] = '\0';
    ht->diff[0] = ht->diff[1] = '.';
    ht->x[0] = x[ht->x0]; ht->y[0] = y[ht->y0];
    ht->x[1] = x[ht->xt]; ht->y[1] = y[ht->yt];
}

```

```

if (bs == 0) {
    if (exact_sc[ht->x[0] - 'A'][ht->y[0] - 'A'] > 0) ht->diff[0] = '.';
    if (ht->x[0] == ht->y[0]) ht->diff[0] = '|';
    } else if (bs == 1)
    ht->y[0] = '-';
    else
    ht->x[0] = '-';

    if (es == 0) {
        if (exact_sc[ht->x[1] - 'A'][ht->y[1] - 'A'] > 0) ht->diff[1] = '.';
        if (ht->x[1] == ht->y[1]) ht->diff[1] = '|';
        } else if (es == 1)
        ht->y[1] = '-';
        else
        ht->x[1] = '-';

        ht->len = 2;
        return;
    }

    /* Define h_line and v_line */
    h_line = (lenx-1) / 2; if (lenx == 1) h_line = -1;
    v_line = (leny-1) / 2; if (leny == 1) v_line = -1;
    /* allocate memory */
    if (! (col = (lnode *) malloc (1*leny * sizeof (lnode))))
        error (1, -1, "couldn't allocate memory for nodes in linear alignment.\n");
    if (! (prevcol = (lnode *) malloc (1*leny * sizeof (lnode))))
        error (1, -1, "couldn't allocate memory for nodes in linear alignment.\n");

    zero.sc[0] = zero.sc[1] = zero.sc[2] = 0.0;
    bad.sc[0] = bad.sc[1] = bad.sc[2] = -1000000.0;
    for (j = 0; j < leny; j++) col[j] = prevcol[j] = bad;

    /* main loop */
    for (i = 0; i < lenx; i++) {
        /* Advance one column and initialize pointers */
        if (ht->type == TANDEM) {
            loop_end = ht->x0 - ht->y0 + i; if (loop_end > leny) loop_end = leny;
        } else loop_end = leny;
        tmp_node = col; col = prevcol; prevcol = tmp_node;
        current = col; left = prevcol;
        diag = &bad; down = &bad;
        if (i == 0) diag = &zero;

        for (j = 0; j < loop_end; j++) {
            /* initialize x0, y0 and s0 */
            if ((i == h_line && j >= v_line) || (i >= h_line && j == v_line)) {
                for (k = 0; k < 3; k++) {
                    current->x0[k] = i;
                    current->y0[k] = j;
                    current->s0[k] = k;
                }
            }
            /* update the scores */
            update = 0;
            if (i > h_line && j > v_line) update = 1;

            /* state 0 : match */
            current->sc[0] = diag->sc[0];
            if (update) {

```

```

current->x0[0] = diag->x0[0];
current->y0[0] = diag->y0[0];
current->s0[0] = diag->s0[0];
}
if (diag->sc[1] > current->sc[0]) {
    current->sc[0] = diag->sc[1];
    if (update) {
        current->x0[0] = diag->x0[1];
        current->y0[0] = diag->y0[1];
        current->s0[0] = diag->s0[1];
    }
}
if (diag->sc[2] > current->sc[0]) {
    current->sc[0] = diag->sc[2];
    if (update) {
        current->x0[0] = diag->x0[2];
        current->y0[0] = diag->y0[2];
        current->s0[0] = diag->s0[2];
    }
}
/* state 1: insert */
current->sc[1] = left->sc[0] - X_GO;
if (update) {
    current->x0[1] = left->x0[0];
    current->y0[1] = left->y0[0];
    current->s0[1] = left->s0[0];
}
if (left->sc[1] - X_GE > current->sc[1]) {
    current->sc[1] = left->sc[1] - X_GE;
    if (update) {
        current->x0[1] = left->x0[1];
        current->y0[1] = left->y0[1];
        current->s0[1] = left->s0[1];
    }
}
/* state 2: delete */
current->sc[2] = down->sc[0] - Y_GO;
if (update) {
    current->x0[2] = down->x0[0];
    current->y0[2] = down->y0[0];
    current->s0[2] = down->s0[0];
}
if (down->sc[2] - Y_GE > current->sc[2]) {
    current->sc[2] = down->sc[2] - Y_GE;
    if (update) {
        current->x0[2] = down->x0[2];
        current->y0[2] = down->y0[2];
        current->s0[2] = down->s0[2];
    }
}
current->sc[0] += exact_sc[x[ht->x0] - 'A'][y[j+ht->y0] - 'A'];
if (i == 0 && j == 0)
    for (k = 0; k < 3; k++)
        if (bs == k) current->sc[k] = zero.sc[k];
        else current->sc[k] = bad.sc[k];
/* Advance pointers for next node */
diag = left; left++; down = current; current++;

```

dp.c

```

}
x0 = down->x0[es] + ht->x0;
y0 = down->y0[es] + ht->y0;
s0 = down->s0[es];
create_hilltop (&bottom);
bottom->type = top->type = ht->type;
bottom->x0 = ht->x0; bottom->y0 = ht->y0;
bottom->xt = x0; bottom->yt = y0;
top->x0 = x0; top->y0 = y0;
top->xt = ht->xt; top->yt = ht->yt;
linear_alignment (x, y, bottom, bs, s0);
linear_alignment (x, y, top, s0, es);
ht->len = bottom->len + top->len - 1;
if (!ht->x = (char *) malloc (ht->len+1))
    error(1, -1, "couldn't allocate memory for alignment string.\n");
if (!ht->y = (char *) malloc (ht->len+1))
    error(1, -1, "couldn't allocate memory for alignment string.\n");
if (!ht->diff = (char *) malloc (ht->len+1))
    error(1, -1, "couldn't allocate memory for alignment string.\n");
for (i = 0; i < bottom->len; i++) {
    ht->x[i] = bottom->x[i];
    ht->y[i] = bottom->y[i];
    ht->diff[i] = bottom->diff[i];
}
for (i = 0; i < top->len; i++) {
    ht->x[i+bottom->len-1] = top->x[i];
    ht->y[i+bottom->len-1] = top->y[i];
    ht->diff[i+bottom->len-1] = top->diff[i];
}
ht->x[ht->len] = ht->y[ht->len] = ht->diff[ht->len] = '\0';
ht->final = 1;
destroy_hilltop (&bottom);
destroy_hilltop (&top);
free (col);
free (prevcol);
}
/*=====
int XL_GO, XL_GE, YL_GO, YL_GE;
void init_long_gaps (lGE, lGO)
{
    XL_GO = lGO;
    XL_GE = lGE;
    YL_GO = lGO;
    YL_GE = lGE;
}
/*=====
hilltop *get_six18_alignment (char x[], char y[], int mode)
/*
 * Dynamic programming routine to match two nucleotide sequences, using
 * square memory, using a six-state mode.
 * Mode is formed by OR'ing the following constants (defined in dp.h):

```

dp.c

```

* ONE_HILLTOP (formerly 0): one hilltop as output.
* SEPARATE_HILLTOPS (formerly 1): separate at long gaps.
* OVERLAP_MODE (0, so you can just ignore it): find best overlapping alignment
* LOCAL_MODE: find (local) best alignment.
*/

```

```

int i, j, best_i = 0, best_j = 0, best_s = DIAG;
int tmp, id_count, sim_count, no_n_count;
int lenx, leny;
double max_score = -1000000.0;
sixnode *diag, *left, *down, *current, *mat, zero;
hilltop *res, *ht;

create_hilltop (&ht);
lenx = strlen (x);
leny = strlen (y);

if ((lenx*leny*sizeof(sixnode)>MAX_FULL*MAX_FULL*sizeof(node)) ||
    mode & LINEAR_SPACE) {
    /* We need a linear memory alignment here (which is slower) */
    call_linear_six18_alignment (x, y, ht, mode & -HILLTOP_MASK);
    if (ht->x[0] == '-' || ht->x0 += 1;
    if (ht->y[0] == '-' || ht->y0 += 1;
    ht->final = 1;
    improve_six18_result (ht, mode & 1);
    return (ht);
}

/* allocate memory and initialize matrix boundaries */
if (!mat = (sixnode *) malloc ((1+lenx*leny * sizeof (sixnode))))
    error(1, -1, "couldn't allocate memory for nodes 1 (%d %d) \n",
        lenx, leny);

```

```

zero.match = zero.ins = zero.del = zero.xgap = zero.ygap = zero.alt = 0;

/* main loop */
for (i = 0; i < lenx; i++) {
    /* Advance one column and initialize pointers */
    current = &mat[i*leny];
    if (i == 0) left = &zero;
    else left = &mat[(i-1)*leny];
    diag = &zero; down = &zero;

    for (j = 0; j < leny; j++) {
        current->match = diag->match;
        if (diag->ins > current->match) current->match = diag->ins;
        if (diag->del > current->match) current->match = diag->del;
        if (diag->xgap-XL_GO > current->match) current->match = diag->xgap-XL_GO;
        if (diag->ygap-YL_GO > current->match) current->match = diag->ygap-YL_GO;
        if (diag->alt-XL_GO > current->match) current->match = diag->alt-XL_GO;
        current->match += exact_sc[x[i]-'A'][y[j]-'A'];

        current->ins = left->match - X_GO;
        if (left->ins - X_GE > current->ins) current->ins = left->ins - X_GE;
        current->del = down->match - Y_GO;
        if (down->del - Y_GE > current->del) current->del = down->del - Y_GE;
        current->xgap = left->match - XL_GO;

```

```

if (left->xgap - XL_GE > current->xgap) current->xgap = left->xgap-XL_GE;
if (left->alt - XL_GE > current->xgap) current->xgap = left->alt - XL_GE;

current->ygap = down->match - YL_GO;
if (down->ygap - YL_GE > current->ygap) current->ygap = down->ygap-YL_GE;
if (down->alt - YL_GE > current->ygap) current->ygap = down->alt - YL_GE;

current->alt = diag->match - YL_GO;
if (diag->alt - YL_GE > current->alt) current->alt = diag->alt - YL_GE;

/* No gap open penalty for first gap (if at edge) */
if (i == 0 || j == 0)
    current->xgap = current->ygap = current->alt = 0;

/* A local alignment may start anywhere */
if ((mode & LOCAL_MODE) &&
    (current->match < 0))
    current->match = 0.0;

```

```

/* Save best state and node */
/* Overlap mode only maximises over the ends of the sequences, local
over the entire matrix; local alignment also has to make sure
to take the shortest sequence which yields the maximal score. */
if ((mode & LOCAL_MODE) &&
    short_allow_equal == ((i+j < best_i+best_j) ||
        ((i+j == best_i+best_j) && (i < best_i))))
#define BEATS(a,b) (((a)>(b)) || (allow_equal && (a)==(b)))
    if (BEATS(current->match, max_score)) {
        max_score = current->match;
        best_i = i; best_j = j; best_s = DIAG;
    }
    if (BEATS(current->ins, max_score)) {
        max_score = current->ins;
        best_i = i; best_j = j; best_s = LEFT;
    }
    if (BEATS(current->del, max_score)) {
        max_score = current->del;
        best_i = i; best_j = j; best_s = DOWN;
    }
    if (BEATS(current->xgap, max_score)) {
        max_score = current->xgap;
        best_i = i; best_j = j; best_s = XGAP;
    }
    if (BEATS(current->ygap, max_score)) {
        max_score = current->ygap;
        best_i = i; best_j = j; best_s = YGAP;
    }
    if (BEATS(current->alt, max_score)) {
        max_score = current->alt;
        best_i = i; best_j = j; best_s = ALT;
    }
#undef BEATS
}
else if (i == lenx-1 || j == leny-1) {
    if (current->match >= max_score) {
        max_score = current->match;
        best_i = i; best_j = j; best_s = DIAG;
    }
    if (current->ins >= max_score) {

```

```

max_score = current->ins;
best_i = i; best_j = j; best_s = LEFT;
}
if (current->del >= max_score) {
max_score = current->del;
best_i = i; best_j = j; best_s = DOWN;
}
if (current->xgap >= max_score) {
max_score = current->xgap;
best_i = i; best_j = j; best_s = XGAP;
}
if (current->ygap >= max_score) {
max_score = current->ygap;
best_i = i; best_j = j; best_s = YGAP;
}
if (current->alt >= max_score) {
max_score = current->alt;
best_i = i; best_j = j; best_s = ALT;
}
}

/* Advance pointers for next node */
if (i > 0) {diag = left; left++;}
down = current; current++;
}

get_six18_match (x, y, mat, best_i, best_j, best_s, ht,
mode & -HILLTOP_MASK);
ht->score = max_score;
improve_six18_result (ht, mode & 1);
free (mat);
return (ht);
}

```

```

/*=====
static void calc_score(hilltop *ht)
/*
* (written by avner, modified by raveh to comply with gap-open convention)
* given the alignment, calculate the score
*/
{
int i;
ht->score=0.;
/* printf("x = %s\n",ht->x,ht->y); */
for (i=0; i< ht->len; i++) {
/* printf("%2f, ",ht->score); */
if (ht->x[i]!='-') {
if (i > 0 && ht->x[i-1] != '-')
ht->score -= X_GO/*-X_GE*/;
else ht->score -= X_GE;
}
else if (ht->y[i]!='-') {
if (i > 0 && ht->y[i-1] != '-')
ht->score -= X_GO/*-X_GE*/;
else ht->score -= X_GE;
}
else ht->score += exact_sc[ht->x[i]-'A'][ht->y[i]-'A'];
}
}

```

dp.c

```

/*=====
void improve_six18_result (hilltop *ht, int mode)
{
int xpos, ypos, i, first;
hilltop *ht2;

if (mode != 0) {
xpos = ht->xt;
ypos = ht->yt;
for (i = strlen (ht->diff)-1; first=1; i >= 0; i--, first=0) {
if (ht->diff[i] == '=') {
if (!first)
/* take align segment right of '=' region */
create_hilltop (&ht2);
copy_hilltop (*ht, ht2);
ht->next = ht2;
free (ht2->x);
free (ht2->y);
free (ht2->diff);
ht2->x = subseq (ht->x, i+1, strlen (ht->x));
ht2->y = subseq (ht->y, i+1, strlen (ht->y));
ht2->diff = subseq (ht->diff, i+1, strlen (ht->diff));
ht2->x0 = xpos+1;
ht2->y0 = ypos+1;
ht2->len = strlen (ht2->diff);
calc_score(ht2);
}
}

for ( ; i>=0 && ht->diff[i] == '='; i--) { /* go to left of '=' region */
if (ht->x[i] != '-') xpos--;
if (ht->y[i] != '-') ypos--;
}
if (i>=0)
/* prepare reminder alignments */
{
i++;
ht->x[i] = ht->y[i] = ht->diff[i] = '\0';
ht->xt = xpos;
ht->yt = ypos;
ht->len = strlen (ht->diff);
}
else if (!first)
/* =====XXXXXXXXX */
{
copy_hilltop (*ht2, ht);
ht->next=ht2->next;
free (ht2->x);
free (ht2->y);
free (ht2->diff);
free(ht2);
}
else
ht->len=0;
}
else
{
if (ht->x[i] != '-') xpos--;
if (ht->y[i] != '-') ypos--;
}
}
}

```

dp.c

```

calc_score(ht);
/* printf("score calculated #2 = %f\n",ht->score);i*/
/*=====*/
/* Extract the best, shortest solution from a six_state model alignment. If
 * mode=0 then we're looking for an overlap alignment (the matrix might be
 * negative); when mode=LOCAL_MODE we're looking for local alignments
 * (any zero in the matrix indicates the end of the backtracking).
 */
void get_six18_match(char x[], char y[], sixnode *mat, int it, int jt, int st,
                    hilltop *ht, int mode)
/* This comment is obsolete!
 *
 * Extract the best solution from a six_state model alignment. If mode=0
 * we want it as one Hilltop. if mode=1, we want it separated at large gaps.
 */
{
    char *nice_x, *nice_y, *nice_diff;
    int last, i, j, loc, end, id_count, sim_count, no_p_count, lenx, leny;
    double tmp;
    sixnode *current, *diag, *left, *down;

    lenx = strlen(x);
    leny = strlen(y);

    end = it + jt;
    if ((nice_x = (char *)malloc(end+2)) == NULL) ||
        (nice_y = (char *)malloc(end+2)) == NULL) ||
        (nice_diff = (char *)malloc(end+2)) == NULL) ||
        error(1, -1, "Can't allocate memory for nice_strings");
    nice_x[end+1] = nice_y[end+1] = nice_diff[end+1] = '\0';
    i = it; j = jt; last = st;

    for (loc = end; /* See test at bottom of loop */; loc--) {
        current = &mat[i * leny + j];
        left = &mat[(i-1) * leny + j];
        down = &mat[i * leny + j-1];
        diag = &mat[(i-1) * leny + j-1];

        if (last == DIAG) {
            nice_x[loc] = x[i];
            nice_y[loc] = y[j];
            if (x[i] == y[j]) nice_diff[loc] = '|';
            else nice_diff[loc] = {exact_sc[x[i]-A][y[j]-A] > 0 ? ' ': ''};
            tmp = current->match - exact_sc[x[i]-A][y[j]-A];
            i--; j--;
            if ((mode & LOCAL_MODE) && fabs(current->match) < EPSILON)
                break;
            if (i >= 0 && j >= 0) {
                if (fabs(diag->match - tmp) < EPSILON) last = DIAG;
                else if (fabs(diag->del - tmp) < EPSILON) last = DOWN;
                else if (fabs(diag->ins - tmp) < EPSILON) last = LEFT;
                else if (fabs(diag->alt - tmp - XL_GO) < EPSILON) last = ALT;
                else if (fabs(diag->xgap - tmp - XL_GO) < EPSILON) last = XGAP;
                else if (fabs(diag->ygap - tmp - XL_GO) < EPSILON) last = YGAP;
                else err("error in back-tracking 1, %d %d\n", i, j);
            }
        }
    }
}

```

```

} else if (last == LEFT) {
    nice_x[loc] = x[i];
    nice_y[loc] = '-';
    nice_diff[loc] = '-';
    tmp = current->ins;
    i--;
    if (i >= 0 && j >= 0) {
        if (fabs(left->match - tmp - XL_GO) < EPSILON) last = DIAG;
        else if (fabs(left->ins - tmp - XL_GO) < EPSILON) last = LEFT;
        else err("error in back-tracking 2, %d %d\n", i, j);
    }
} else if (last == DOWN) {
    nice_x[loc] = '-';
    nice_y[loc] = y[j];
    nice_diff[loc] = '-';
    tmp = current->del;
    j--;
    if (i >= 0 && j >= 0) {
        if (fabs(down->del - tmp - YL_GE) < EPSILON) last = DOWN;
        else if (fabs(down->match - tmp - Y_GO) < EPSILON) last = DIAG;
        else err("error in back-tracking 3, %d %d\n", i, j);
    }
} else if (last == XGAP) {
    nice_x[loc] = x[i];
    nice_y[loc] = '-';
    nice_diff[loc] = '=';
    tmp = current->xgap;
    i--;
    if (i >= 0 && j >= 0) {
        if (fabs(left->xgap - tmp - XL_GE) < EPSILON) last = XGAP;
        else if (fabs(left->alt - tmp - XL_GE) < EPSILON) last = ALT;
        else if (fabs(left->match - tmp - XL_GO) < EPSILON) last = DIAG;
        else err("error in back-tracking 4, %d %d\n", i, j);
    }
} else if (last == YGAP) {
    nice_x[loc] = '-';
    nice_y[loc] = y[j];
    nice_diff[loc] = '=';
    tmp = current->ygap;
    j--;
    if (i >= 0 && j >= 0) {
        if (fabs(down->ygap - tmp - YL_GE) < EPSILON) last = YGAP;
        else if (fabs(down->alt - tmp - YL_GE) < EPSILON) last = ALT;
        else if (fabs(down->match - tmp - YL_GO) < EPSILON) last = DIAG;
        else err("error in back-tracking 5, %d %d\n", i, j);
    }
} else if (last == ALT) {
    nice_x[loc] = x[i];
    nice_y[loc] = y[j];
    nice_diff[loc] = '-';
    tmp = current->alt;
    i--; j--;
    if (i >= 0 && j >= 0) {
        if (fabs(diag->alt - tmp - YL_GE) < EPSILON) last = ALT;
        else if (fabs(diag->match - tmp - YL_GO) < EPSILON) last = DIAG;
        else err("error in back-tracking 6, %d %d\n", i, j);
    }
}
if (i < 0 || j < 0)

```



```

break;
}
if (last != DOWN && last != YGAP) i++;
if (last != LEFT && last != XGAP) j++;

ht->final = 1;
ht->len = strlen (nice_y[loc]);
ht->x = subseq (nice_x[loc], 0, ht->len);
ht->y = subseq (nice_y[loc], 0, ht->len);
ht->diff = subseq (nice_diff[loc], 0, ht->len);
ht->xt = it;
ht->yt = jt;
ht->x0 = i; if (ht->x[0] == '-') ht->x0 += 1;
ht->y0 = j; if (ht->y[0] == '-') ht->y0 += 1;
free (nice_x);
free (nice_y);
free (nice_diff);

id_count = sim_count = no_n_count = 0;
for (i = 0; i < strlen (ht->diff); i++) {
    id_count += (ht->diff[i] == '|');
    if (ht->x[i] != '-' && ht->y[i] != '-')
        sim_count += (exact_sc[ht->x[i]]-'A'][ht->y[i]]-'A'] > 0);
    if (ht->x[i] != 'N' && ht->y[i] != 'N') no_n_count++;
}
ht->id_percent = 100*(double)(id_count) / (double)(no_n_count);
ht->sim_percent = 100*(double)(sim_count) / (double)(no_n_count);
}

/*=====
void call_linear_six18_alignment (char x[], char y[], hilltop *ht, int mode)
/*
* Dynamic programming routine for first stage of matching two nucleotide
* sequences, using linear memory, in the 6-state, 18-transition model.
* See get_six18_match for explanation of mode (Hilltops options are
* NOT implemented here, but rather in get_six18_match).
*/
{
    int i, j, k, x0, y0, s0, xt, yt, st;
    int lenx, leny, loop_end, h_line, v_line, update;
    lsixnode *diag, *left, *down, *current, *tmp_node;
    lsixnode *col, *prevcol, zero, bad;
    hilltop *bottom, *top;
    double max_score = -1000000.0;

    lenx = strlen (x);
    leny = strlen (y);

```

```

/* allocate memory */
if (!col = (lsixnode *) malloc ((1+leny * sizeof (lsixnode))))
    error(1, -1, "couldn't allocate memory for nodes in linear alignment.\n");
if (!prevcol = (lsixnode *) malloc ((1+leny * sizeof (lsixnode))))
    error(1, -1, "couldn't allocate memory for nodes in linear alignment.\n");

zero.sc[0] = zero.sc[1] = zero.sc[2] = zero.sc[3] = zero.sc[4] = zero.sc[5] = 0.0;
zero.x0[0] = zero.x0[1] = zero.x0[2] = zero.x0[3] = zero.x0[4] = zero.x0[5] = 0.0;
zero.y0[0] = zero.y0[1] = zero.y0[2] = zero.y0[3] = zero.y0[4] = zero.y0[5] = 0.0;
zero.s0[0] = zero.s0[1] = zero.s0[2] = zero.s0[3] = zero.s0[4] = zero.s0[5] = 0.0;

```

dp.c

```

bad.sc[0] = bad.sc[1] = bad.sc[2] = bad.sc[3] = bad.sc[4] = bad.sc[5] = -1000000;
for (j = 0; j < leny; j++) col[j] = prevcol[j] = zero;

/* main loop */
for (i = 0; i < lenx; i++) {
    /* Advance one column and initialize pointers */
    tmp_node = col; col = prevcol; prevcol = tmp_node;
    current = col; left = prevcol;
    diag = &zero; down = &zero;

    for (j = 0; j < leny; j++) {
        /* state 0 : match */
        current->sc[0] = diag->sc[0];
        current->x0[0] = diag->x0[0];
        current->y0[0] = diag->y0[0];
        current->s0[0] = diag->s0[0];
        if (diag->sc[1] > current->sc[0]) {
            current->sc[0] = diag->sc[1];
            current->x0[0] = diag->x0[1];
            current->y0[0] = diag->y0[1];
            current->s0[0] = diag->s0[1];
        }
        if (diag->sc[2] > current->sc[0]) {
            current->sc[0] = diag->sc[2];
            current->x0[0] = diag->x0[2];
            current->y0[0] = diag->y0[2];
            current->s0[0] = diag->s0[2];
        }
        if (diag->sc[3] > current->sc[0]) {
            current->sc[0] = diag->sc[3] - XL_GO;
            current->x0[0] = diag->x0[3];
            current->y0[0] = diag->y0[3];
            current->s0[0] = diag->s0[3];
        }
        if (diag->sc[4] > current->sc[0]) {
            current->sc[0] = diag->sc[4] - XL_GO;
            current->x0[0] = diag->x0[4];
            current->y0[0] = diag->y0[4];
            current->s0[0] = diag->s0[4];
        }
        if (diag->sc[5] > current->sc[0]) {
            current->sc[0] = diag->sc[5] - XL_GO;
            current->x0[0] = diag->x0[5];
            current->y0[0] = diag->y0[5];
            current->s0[0] = diag->s0[5];
        }
        current->sc[0] += exact_sc[(x[i]+ht->x0)-(y[j]+ht->y0)]-'A'];
    }
    /* A local alignment may start anywhere, but can't have -ve
    scores */
    if ((mode & LOCAL_MODE) && (current->sc[0] < 0)) {
        current->sc[0] = 0.0;
        current->x0[0] = i;
        current->y0[0] = j;
    }
}

/* State 1: insert */
current->sc[1] = left->sc[0] - X_GO;
current->x0[1] = left->x0[0];

```

dp.c

A-64

```

current->y0[1] = left->y0[0];
current->s0[1] = left->s0[0];
if (left->sc[1] - X_GE > current->sc[1]) {
    current->sc[1] = left->sc[1] - X_GE;
    current->x0[1] = left->x0[1];
    current->y0[1] = left->y0[1];
    current->s0[1] = left->s0[1];
}

/* State 2: delete */
current->sc[2] = down->sc[0] - Y_GO;
current->x0[2] = down->x0[0];
current->y0[2] = down->y0[0];
current->s0[2] = down->s0[0];
if (down->sc[2] - Y_GE > current->sc[2]) {
    current->sc[2] = down->sc[2] - Y_GE;
    current->x0[2] = down->x0[2];
    current->y0[2] = down->y0[2];
    current->s0[2] = down->s0[2];
}

/* state 3: X-gap */
current->sc[3] = left->sc[0] - XL_GO;
current->x0[3] = left->x0[0];
current->y0[3] = left->y0[0];
current->s0[3] = left->s0[0];
if (left->sc[3] - XL_GE > current->sc[3]) {
    current->sc[3] = left->sc[3] - XL_GE;
    current->x0[3] = left->x0[3];
    current->y0[3] = left->y0[3];
    current->s0[3] = left->s0[3];
}
if (left->sc[5] - XL_GE > current->sc[3]) {
    current->sc[3] = left->sc[5] - XL_GE;
    current->x0[3] = left->x0[5];
    current->y0[3] = left->y0[5];
    current->s0[3] = left->s0[5];
}

/* state 4: Y-gap */
current->sc[4] = down->sc[0] - YL_GO;
current->x0[4] = down->x0[0];
current->y0[4] = down->y0[0];
current->s0[4] = down->s0[0];
if (down->sc[4] - YL_GE > current->sc[4]) {
    current->sc[4] = down->sc[4] - YL_GE;
    current->x0[4] = down->x0[4];
    current->y0[4] = down->y0[4];
    current->s0[4] = down->s0[4];
}
if (down->sc[5] - YL_GE > current->sc[4]) {
    current->sc[4] = down->sc[5] - YL_GE;
    current->x0[4] = down->x0[5];
    current->y0[4] = down->y0[5];
    current->s0[4] = down->s0[5];
}

/* State 5: Alternate */
current->sc[5] = diag->sc[0] - YL_GO;
current->x0[5] = diag->x0[0];

```

dp.c

```

current->y0[5] = diag->y0[0];
current->s0[5] = diag->s0[0];
if (diag->sc[5] - YL_GE > current->sc[5]) {
    current->sc[5] = diag->sc[5] - YL_GE;
    current->x0[5] = diag->x0[5];
    current->y0[5] = diag->y0[5];
    current->s0[5] = diag->s0[5];
}

/* initialize x0, y0 and s0 */
if ((i == 0) || (j == 0)) {
    for (k = 0; k < 6; k++) {
        if (i != 0 && (k == 1 || k == 3)) continue;
        if (j != 0 && (k == 2 || k == 4)) continue;
        current->x0[k] = i;
        current->y0[k] = j;
        current->s0[k] = k;
    }
}

/* Save best end point */
/* See comment in get_six18_alignment() for the difference in
   modes */
if (mode & LOCAL_MODE) {
    short allow_equal = ((i+j < xt+yt) ||
                        ((i+j == xt+yt) && (i < xt)));
    #define BEATS(a, b) (((a)>(b)) || (allow_equal && (a)==(b)))
    for (k=0; k<6; k++)
        if (BEATS(current->sc[k], max_score)) {
            max_score = current->sc[k];
            x0 = current->x0[k];
            y0 = current->y0[k];
            s0 = current->s0[k]; /* Will always be MATCH for local
                               alignment */
            xt = i;
            yt = j;
            st = k;
        }
    #undef BEATS
}

else if (i == lenx-1 || j == leny-1)
    for (k = 0; k < 6; k++)
        if (current->sc[k] >= max_score) {
            max_score = current->sc[k];
            x0 = current->x0[k];
            y0 = current->y0[k];
            s0 = current->s0[k];
            xt = i;
            yt = j;
            st = k;
        }

/* Advance pointers for next node */
diag = left; left++; down = current; current++;
}
free (col);
free (prevcol);

```

dp.c

```

ht->x0 = x0;
ht->y0 = y0;
ht->xt = xt;
ht->yt = yt;
linear_six18_alignment (x, y, ht, s0, st, mode);
}

/*=====
void linear_six18_alignment (char x[], char y[], hilltop *ht, int bs, int es,
                           int mode)
*/
* Recursive dynamic programming routine to match two nucleotide sequences,
* using linear memory, in the 6-state, 18-transition model.
* mode can be OVERLAP_MODE (0), or LOCAL_MODE, but makes no difference here.
*/
{
    int i, j, k, x0, y0, s0;
    int lenx, leny, loop_end, h_line, v_line, update;
    lsixnode *diag, *left, *down, *current, *tmp_node;
    lsixnode *col, *prevcol, zero, bad;
    hilltop *bottom, *top;

    /*printf ("In linear_alignment x0=%d, xt=%d, y0=%d, yt=%d, bs=%d, es=%d\n",
    ht->x0, ht->xt, ht->y0, ht->yt, bs, es); */

    lenx = ht->xt - ht->x0 + 1;
    leny = ht->yt - ht->y0 + 1;
    /* First check if this is the end of the recursion */
    if (lenx <= 2 && leny <= 2) {
        if (!ht->x = (char *) malloc (3))
            error(1, -1, "couldn't allocate memory for alignment string.\n");
        if (!ht->y = (char *) malloc (3))
            error(1, -1, "couldn't allocate memory for alignment string.\n");
        if (!ht->diff = (char *) malloc (3))
            error(1, -1, "couldn't allocate memory for alignment string.\n");
        ht->x[2] = ht->y[2] = ht->diff[2] = '\0';
        ht->diff[0] = ht->diff[1] = ' ';
        ht->x[0] = x[ht->x0]; ht->y[0] = y[ht->y0];
        ht->x[1] = x[ht->xt]; ht->y[1] = y[ht->yt];

        if (bs == 0) {
            if (exact_sc[ht->x[0] - 'A'][ht->y[0] - 'A'] > 0) ht->diff[0] = ' ';
            if (ht->x[0] == ht->y[0]) ht->diff[0] = '|';
        } else if (bs == 1)
            ht->y[0] = '-';
        else if (bs == 2)
            ht->x[0] = '-';
        else if (bs == 3) {
            ht->y[0] = '-';
            ht->diff[0] = '=';
        } else if (bs == 4) {
            ht->x[0] = '-';
            ht->diff[0] = '=';
        } else
            ht->diff[0] = '=';
        ht->diff[1] = '=';
    }
    if (es == 0) {
        if (exact_sc[ht->x[1] - 'A'][ht->y[1] - 'A'] > 0) ht->diff[1] = ' ';
    }
}

```

dp.c

```

if (ht->x[1] == ht->y[1]) ht->diff[1] = '|';
} else if (es == 1)
    ht->y[1] = '-';
else if (es == 2)
    ht->x[1] = '-';
else if (es == 3) {
    ht->y[1] = '-';
    ht->diff[1] = '=';
} else if (es == 4) {
    ht->x[1] = '-';
    ht->diff[1] = '=';
} else
    ht->diff[1] = '=';
ht->len = 2;
return;
}

/* Define h_line and v_line */
h_line = (lenx-1) / 2; if (lenx == 1) h_line = -1;
v_line = (leny-1) / 2; if (leny == 1) v_line = -1;
/* allocate memory */
if (!col = (lsixnode *) malloc ((1+leny * sizeof (lsixnode))))
    error(1, -1, "couldn't allocate memory for nodes in linear alignment.\n");
if (!prevcol = (lsixnode *) malloc ((1+leny * sizeof (lsixnode))))
    error(1, -1, "couldn't allocate memory for nodes in linear alignment.\n");
zero.sc[0] = zero.sc[1] = zero.sc[2] = zero.sc[3] = zero.sc[4] = zero.sc[5] = 0.0;
bad.sc[0] = bad.sc[1] = bad.sc[2] = bad.sc[3] = bad.sc[4] = bad.sc[5] = -1000000;
for (j = 0; j < leny; j++) col[j] = prevcol[j] = bad;

/* main loop */
for (i = 0; i < lenx; i++) {
    /* Advance one column and initialize pointers */
    tmp_node = col; col = prevcol; prevcol = tmp_node;
    current = col; left = prevcol;
    diag = &bad; down = &zero;
    if (i == 0) diag = &zero;
    for (j = 0; j < leny; j++) {
        /* initialize x0, y0 and s0 */
        if ((i == h_line && j >= v_line) || (i >= h_line && j == v_line)) {
            for (k = 0; k < 6; k++) {
                current->x0[k] = i;
                current->y0[k] = j;
                current->s0[k] = k;
            }
        }
        /* update the scores */
        update = 0;
        if (i > h_line && j > v_line) update = 1;
        /* state 0 : match */
        current->sc[0] = diag->sc[0];
        if (update) {
            current->x0[0] = diag->x0[0];
            current->y0[0] = diag->y0[0];
            current->s0[0] = diag->s0[0];
        }
        if (diag->sc[1] > current->sc[0]) {

```

dp.c

A-66

```

current->sc[0] = diag->sc[1];
if (update) {
    current->x0[0] = diag->x0[1];
    current->y0[0] = diag->y0[1];
    current->s0[0] = diag->s0[1];
}
}
if (diag->sc[2] > current->sc[0]) {
    current->sc[0] = diag->sc[2];
    if (update) {
        current->x0[0] = diag->x0[2];
        current->y0[0] = diag->y0[2];
        current->s0[0] = diag->s0[2];
    }
}
}
if (diag->sc[3] - XL_GO > current->sc[0]) {
    current->sc[0] = diag->sc[3] - XL_GO;
    if (update) {
        current->x0[0] = diag->x0[3];
        current->y0[0] = diag->y0[3];
        current->s0[0] = diag->s0[3];
    }
}
}
if (diag->sc[4] - XL_GO > current->sc[0]) {
    current->sc[0] = diag->sc[4] - XL_GO;
    if (update) {
        current->x0[0] = diag->x0[4];
        current->y0[0] = diag->y0[4];
        current->s0[0] = diag->s0[4];
    }
}
}
if (diag->sc[5] - XL_GO > current->sc[0]) {
    current->sc[0] = diag->sc[5] - XL_GO;
    if (update) {
        current->x0[0] = diag->x0[5];
        current->y0[0] = diag->y0[5];
        current->s0[0] = diag->s0[5];
    }
}
}
current->sc[0] += exact_sc[x[i+ht->x0] - 'A'][y[j+ht->y0] - 'A'];

/* State 1: insert */
current->sc[1] = left->sc[0] - X_GO;
if (update) {
    current->x0[1] = left->x0[0];
    current->y0[1] = left->y0[0];
    current->s0[1] = left->s0[0];
}
}
if (left->sc[1] - X_GE > current->sc[1]) {
    current->sc[1] = left->sc[1] - X_GE;
    if (update) {
        current->x0[1] = left->x0[1];
        current->y0[1] = left->y0[1];
        current->s0[1] = left->s0[1];
    }
}
}
/* State 2: delete */
current->sc[2] = down->sc[0] - Y_GO;

```

dp.c

```

if (update) {
    current->x0[2] = down->x0[0];
    current->y0[2] = down->y0[0];
    current->s0[2] = down->s0[0];
}
}
if (down->sc[2] - Y_GE > current->sc[2]) {
    current->sc[2] = down->sc[2] - Y_GE;
    if (update) {
        current->x0[2] = down->x0[2];
        current->y0[2] = down->y0[2];
        current->s0[2] = down->s0[2];
    }
}
}
/* state 3: X-gap */
current->sc[3] = left->sc[0] - XL_GO;
if (update) {
    current->x0[3] = left->x0[0];
    current->y0[3] = left->y0[0];
    current->s0[3] = left->s0[0];
}
}
if (left->sc[3] - XL_GE > current->sc[3]) {
    current->sc[3] = left->sc[3] - XL_GE;
    if (update) {
        current->x0[3] = left->x0[3];
        current->y0[3] = left->y0[3];
        current->s0[3] = left->s0[3];
    }
}
}
if (left->sc[5] - XL_GE > current->sc[3]) {
    current->sc[3] = left->sc[5] - XL_GE;
    if (update) {
        current->x0[3] = left->x0[5];
        current->y0[3] = left->y0[5];
        current->s0[3] = left->s0[5];
    }
}
}
/* state 4: Y-gap */
current->sc[4] = down->sc[0] - YL_GO;
if (update) {
    current->x0[4] = down->x0[0];
    current->y0[4] = down->y0[0];
    current->s0[4] = down->s0[0];
}
}
if (down->sc[4] - YL_GE > current->sc[4]) {
    current->sc[4] = down->sc[4] - YL_GE;
    if (update) {
        current->x0[4] = down->x0[4];
        current->y0[4] = down->y0[4];
        current->s0[4] = down->s0[4];
    }
}
}
if (down->sc[5] - YL_GE > current->sc[4]) {
    current->sc[4] = down->sc[5] - YL_GE;
    if (update) {
        current->x0[4] = down->x0[5];
        current->y0[4] = down->y0[5];
        current->s0[4] = down->s0[5];
    }
}
}

```

dp.c

A-67

```

)
/* State 5: Alternate */
current->sc[5] = diag->sc[0] - YL_GO;
if (update) {
    current->x0[5] = diag->x0[0];
    current->y0[5] = diag->y0[0];
    current->s0[5] = diag->s0[0];
}
if (diag->sc[5] - YL_GE > current->sc[5]) {
    current->sc[5] = diag->sc[5] - YL_GE;
    if (update) {
        current->x0[5] = diag->x0[5];
        current->y0[5] = diag->y0[5];
        current->s0[5] = diag->s0[5];
    }
}
if (i == 0 && j == 0)
    for (k = 0; k < 6; k++)
        if (bs == k) current->sc[k] = zero.sc[k];
        else current->sc[k] = bad.sc[k];
/* Advance pointers for next node */
diag = left; left++; down = current; current++;
}
x0 = down->x0[es] + ht->x0;
y0 = down->y0[es] + ht->y0;
s0 = down->s0[es];
create_hilltop (&bottom);
create_hilltop (&top);
bottom->type = top->type = ht->type;
bottom->x0 = ht->x0; bottom->y0 = ht->y0;
bottom->xt = x0; bottom->yt = y0;
top->x0 = x0; top->y0 = y0;
top->xt = ht->xt; top->yt = ht->yt;
linear_six18_alignment (x, y, bottom, bs, s0, mode);
linear_six18_alignment (x, y, top, s0, es, mode);
ht->len = bottom->len + top->len - 1;
if (! (ht->x = (char *) malloc (ht->len+1)))
    error(1, -1, "couldn't allocate memory for alignment string.\n");
if (! (ht->y = (char *) malloc (ht->len+1)))
    error(1, -1, "couldn't allocate memory for alignment string.\n");
if (! (ht->diff = (char *) malloc (ht->len+1)))
    error(1, -1, "couldn't allocate memory for alignment string.\n");
for (i = 0; i < bottom->len; i++) {
    ht->x[i] = bottom->x[i];
    ht->y[i] = bottom->y[i];
    ht->diff[i] = bottom->diff[i];
}
for (i = 0; i < top->len; i++) {
    ht->x[i+bottom->len-1] = top->x[i];
    ht->y[i+bottom->len-1] = top->y[i];
    ht->diff[i+bottom->len-1] = top->diff[i];
}
ht->x[ht->len] = ht->y[ht->len] = ht->diff[ht->len] = '\0';
ht->final = 1;

```

dp.c

```

destroy_hilltop (&bottom);
destroy_hilltop (&top);
free (col);
free (prevcol);
}
/* ===== */
hilltop *band_sw (char x[], int x0, int y0, int width)
/*
 * Dynamic programming routine. Returns the best overlap alignment between
 * it's two input sequences in a band.
 */
{
    int i, j, best_i = 0, best_j = 0, tmp, id_count, sim_count, no_n_count;
    int lenx, leny, mode, loop_end, band, real_j;
    double max_score = -100000.0;
    node *diag, *left, *down, *current, *mat, zero, bad;
    hilltop *ht;

    create_hilltop (&ht);
    lenx = strlen (x);
    leny = strlen (y);
    if (lenx == 0 || leny == 0)
        error (1, -1, "Sequence of length zero in band_sw\n");
    band = 2*width + 1;

    /* allocate memory and initialize matrix boundaries */
    if (! (mat = (node *) malloc (1+lenx*band * sizeof (node))))
        error (1, -1, "couldn't allocate memory for band nodes 1 (%d %d).\n",
            lenx, leny);

    zero.match = zero.ins = zero.del = 0;
    bad.match = bad.ins = bad.del = -100000;

    /* main loop */
    for (i = 0; i < lenx; i++) {
        for (real_j = 0; real_j < band; real_j++) {
            j = real_j + i - x0 + y0 - width;
            if (j >= leny || j < 0) continue;

            /* Set pointers */
            current = &mat[i*band + real_j];
            if (i == 0 || j == 0) diag = &zero;
            else diag = &mat[(i-1)*band + real_j];

            if (i == 0) left = &zero;
            else if (real_j == band - 1) left = &bad;
            else left = &mat[(i-1)*band + real_j+1];

            if (j == 0) down = &zero;
            else if (real_j == 0) down = &bad;
            else down = &mat[i*band + real_j-1];

            /* Calculate node */
            current->match = diag->match;
            if (diag->ins > current->match) current->match = diag->ins;
            if (diag->del > current->match) current->match = diag->del;
            current->match += exact_sc[x[i] - 'A'][y[j] - 'A'];

```

dp.c

```

current->ins = left->match - X_GO;
if (left->ins - X_GE > current->ins) current->ins = left->ins - X_GE;

current->del = down->match - Y_GO;
if (down->del - Y_GE > current->del) current->del = down->del - Y_GE;

/* Save best node */
if (i == lenx-1 || j == leny-1)
    if ((current->match) >= max_score) {
        max_score = current->match;
        best_i = i;
        best_j = j;
    }
}

get_band_match (x, y, x0, y0, width, mat, best_i, best_j, ht);
free (mat);
return (ht);
}

/* Extract the best solution */
void get_band_match (char x[], char y[], int x0, int y0, int width, node *mat,
/* ===== */
char *nice_x, *nice_y, *nice_diff;
int last, i, j, loc, end, id_count, sim_count, no_n_count, lenx, leny;
int real_j, band;
double tmp;
node *current, *diag, *left, *down, *bad;

lenx = strlen (x);
leny = strlen (y);
bad_match = bad.ins = bad.del = -100000;
band = 2*width + 1;

end = it + jt;
if (((nice_x = (char *)malloc (end+2)) == NULL) ||
    ((nice_y = (char *)malloc (end+2)) == NULL) ||
    ((nice_diff = (char *)malloc (end+2)) == NULL))
    error (1, -1, "Can't allocate memory for nice_strings");
nice_x[end+1] = nice_y[end+1] = nice_diff[end+1] = '\0';

i = it; j = jt;
last = DIAG;

for (loc = end; last != NONE; loc--) {
    real_j = j - i + x0 - y0 + width;

    current = &mat[i * band + real_j];
    if (real_j < band-1)
        left = &mat[(i-1)*band + real_j+1];
    else left = &bad;
    if (real_j > 0)
        down = &mat[i * band + real_j-1];
    else down = &bad;
    diag = &mat[(i-1)*band + real_j];

    if (last == DIAG) {

```

dp.c

```

nice_x[loc] = x[i];
nice_y[loc] = y[j];
if (x[i] == y[j]) nice_diff[loc] = '|';
else nice_diff[loc] = (exact_sc[x[i]-'A'][y[j]-'A'] > 0) ? ' ' : '|';
tmp = current->match - exact_sc[x[i]-'A'][y[j]-'A'];
if (i == 1 || j == 1) last = NONE;
else {
    if (fabs(diag->match - tmp) < EPSILON) last = DIAG;
    else if (fabs(diag->del - tmp) < EPSILON) last = DOWN;
    else if (fabs(diag->ins - tmp) < EPSILON) last = LEFT;
    else
        err ("error in band back-tracking 1, %d %d\n", i, j);
} if (last == NONE) break;
i--; j--;
} else if (last == LEFT) {
    nice_x[loc] = x[i];
    nice_y[loc] = '-';
    nice_diff[loc] = '|';
    if (i == 1 || j == 1) last = NONE;
    else {
        tmp = current->ins;
        if (fabs(left->match - tmp-X_GO) < EPSILON) last = DIAG;
        else if (fabs(left->ins - tmp-X_GE) < EPSILON) last = LEFT;
        else printf ("error in band back-tracking 2, %d %d\n", i, j);
    } i--;
    } else if (last == DOWN) {
        nice_x[loc] = '-';
        nice_y[loc] = y[j];
        nice_diff[loc] = '|';
        if (i == 1 || j == 1) last = NONE;
        else {
            tmp = current->del;
            if (fabs(down->del - tmp-Y_GO) < EPSILON) last = DOWN;
            else if (fabs(down->match - tmp-Y_GE) < EPSILON) last = DIAG;
            else printf ("error in band back-tracking 3, %d %d\n", i, j);
        } j--;
    }
}

ht->final = 1;
ht->len = strlen (nice_y[loc]);
ht->x = subseq (nice_x[loc], 0, ht->len);
ht->y = subseq (nice_y[loc], 0, ht->len);
ht->diff = subseq (nice_diff[loc], 0, ht->len);
ht->xt = it;
ht->yt = jt;
ht->x0 = i;
ht->y0 = j;
destroy_hilltop (&ht);
free (nice_x);
free (nice_y);
free (nice_diff);
}

void calc_id_sim(char* str1, char* str2, int start1, int start2, char *align_str,
double *id_percent, double *sim_percent){

    int id_count, sim_count, no_n_count, i, i1, i2;

    id_count = sim_count = no_n_count = 0;

```

dp.c

```
for (il=startl,i = 0; i < strlen (align_str);i++) {
    id_count += (strl[il] == align_str[i]) && (strl[il] == 'A' || strl[il] == '
    ' ||
        if (strl[il] != '-' && align_str[i] != '-')
            sim_count += (exact_sc(strl[il]-'A')(align_str[i]-'A') > 0);
        if (strl[il] != 'N' && align_str[i] != 'N') no_n_count++;
        il += (align_str[i] < 'a') || (align_str[i] > 'z');
    }
    *id_percent = 100*(double)(id_count) / (double)(no_n_count);
    *sim_percent = 100*(double)(sim_count) / (double)(no_n_count);
}
```



```

/* $Log: error.c,v $
 * Revision 1.6 1998/04/05 06:52:06 eval
 * Adding a call to fflush(stdout) in function err before the exit
 *
 * Revision 1.5 1998/02/22 17:11:15 ariels
 * Add "delimited" one-line reports.
 *
 * Revision 1.4 1998/01/20 09:32:17 ariels
 * Add treatment of "error-lists" for more uniform handling of errors.
 *
 * Revision 1.3 1998/01/08 13:20:23 ariels
 * Use **correct** routines for varargs error reporting (previously worked
 * fortuitously on alphas because of compiler quirks).
 *
 * Revision 1.2 1997/11/30 07:56:54 ariels
 * Added ChangeLog comment and static rcsid string.
 *
 */

```

```
static char rcsid[] = "$Id: error.c,v 1.6 1998/04/05 06:52:06 eval Exp $";
```

```
#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>
#include <string.h>

```

```
#include "error.h"
```

```
/* Maximal size for stored errors/warnings */
#define MAX_ERR 1024

```

```
void err(char *format, ...)
```

```
{
    va_list args;
    fflush(stdout);
    va_start(args, format);
    vfprintf(stderr, format, args);
    va_end(args);
    puts('\n', stderr);
    exit(1);
}

```

```
int ret_err(char *format, ...)
```

```
{
    va_list args;
    va_start(args, format);
    vfprintf(stderr, format, args);
    va_end(args);
    puts('\n', stderr);
    return 0;
}

```

```
static err_list errors = NULL, warnings = NULL;
```

```
static int raised(err_list e)
{

```

error.c

```

    return (e != NULL);
}

static void record(err_list *e, char *fmt, va_list args)
{
    err_list new, p;
    char rec_buf[MAX_ERR];

    vsprintf(rec_buf, fmt, args);
    if ((new = malloc(sizeof(struct err_list_s))) == NULL)
        err("record: Memory failure for %d bytes", sizeof(struct err_list_s));
    if ((new->msg = strdup(rec_buf)) == NULL) {
        free(new);
        err("record: Memory failure for another %d bytes", strlen(rec_buf)+1);
    }
    new->next = NULL;
    if (*e) {
        for(p=e; p->next; p=p->next)
            ;
        /* p now points to currently last node */
        p->next = new;
    }
    else
        *e = new;
}

static void report(err_list e, FILE *fp, char *prefix)
{
    while (e) {
        fprintf(fp, "%s%s\n", prefix, e->msg);
        e=e->next;
    }
}

static void delim_report(err_list e, FILE *fp, char *delim)
{
    short first=1;
    while (e) {
        fprintf(fp, "%s%s", first ? "" : delim, e->msg);
        first = 0;
        e=e->next;
    }
}

static void clear(err_list *e)
{
    err_list p,q;
    for(p=*e; p; p=q) {
        q = p->next;
        free(p->msg);
        free(p);
    }
    *e = NULL;
}

void record_warn(char *fmt, ...)
{
    va_list args;

```

error.c

```
va_start(args, fmt);
record(&warnings, fmt, args);
va_end(args);
}
void record_err(char *fmt, ...)
{
    va_list args;
    va_start(args, fmt);
    record(&errors, fmt, args);
    va_end(args);
}

int warned(void)
{
    return raised(warnings);
}

int erred(void)
{
    return raised(errors);
}

void report_warn(FILE *fp, char *prefix)
{
    report(warnings, fp, prefix);
}

void report_warn_oneline(FILE *fp, char *delim)
{
    delim_report(warnings, fp, delim);
}

void report_err(FILE *fp, char *prefix)
{
    report(errors, fp, prefix);
}

void report_err_oneline(FILE *fp, char *delim)
{
    delim_report(errors, fp, delim);
}

void clear_warn(void)
{
    clear(&warnings);
}

void clear_err(void)
{
    clear(&errors);
}
```

```
/* $Log: est_table.c,v $
* Revision 1.23 1998/06/29 12:16:14 avner
* added function 'est_table__next_in_variant'.
*
* Revision 1.22 1998/06/15 12:13:20 eyal
* Add support for hyper_edge_node list for all the ests, and function which che
ck it
*
* Revision 1.21 1998/05/12 11:02:20 eyal
* Fix wrong printf in verify_path
*
* Revision 1.20 1998/05/11 11:53:30 eyal
* Add function est_table__verify_est_path
*
* Revision 1.19 1998/05/04 12:35:59 eyal
* Add field and get/set functions component
*
* Revision 1.18 1998/04/23 17:57:57 avner
* make second parameter of 'trim_head' to be the length of the cut, rather than
\the absolute place of cut.
*
* Revision 1.17 1998/04/21 13:46:21 eyal
* Add functions est_table__cut_tail and est_table__cut_head
*
* Revision 1.16 1998/04/18 22:19:45 avner
* changes due to the different way in which we deal with the 'dirty' fields.
* new semantics:
* 'original_seq' = sequence read in the input file, minus the trimming of pol
ya
* 'clean_seq' = original_seq minus quality and polyN stuff
*
* Revision 1.15 1998/04/13 10:36:37 eyal
* 1. change feild node_path from short to int.
* 2. add function est_table_switch_node_path_hyper_edge
*
* Revision 1.14 1998/04/05 06:51:08 eyal
* Adding code for freeing hyper_edge
*
* Revision 1.13 1998/03/30 09:10:21 eyal
* Add get/set methods for field hyper_edge_len
*
* Revision 1.12 1998/03/29 20:42:30 avner
* continue uncleaning and clengeth work. add hyper-graph support methods.
*
* Revision 1.11 1998/03/17 16:36:30 avner
* support the recording of tails (suspected in being dirty) of ests.
* start supporting usage of clone-length information.
*
* Revision 1.10 1998/03/08 21:14:13 avner
* add functions 'est_table__mark_alignment_range',
* 'est_table__get_start_in_cons',
* 'est_table__get_end_in_cons' and
* 'est_table__clone_name'
* for the use of clone-length prediction mechanism.
*
* Revision 1.9 1998/03/05 09:31:50 avner
* add function 'est_table__strand' that will tell what direction we took
* the sequence to be. This is for the assembly to use, not the flipper of
* course.
*

```

```

* Revision 1.8 1998/02/22 17:19:11 ariels
* Make "Inconsistent directions" warning go through proper channels
* (record_warn())
*
* Revision 1.7 1998/02/11 16:23:55 ariels
* Support 'contig' field for (separate) full-cluster flipper program.
*
* Revision 1.6 1998/01/21 07:52:49 ariels
* Put err() in scope of prototype (add #include "error.h")
*
* Revision 1.5 1998/01/14 10:05:40 eyal
* Add support for rna_test_mode (set get for test_case field).
*
* Revision 1.4 1998/01/11 13:55:15 ariels
* Fix error reporting to use err()
*
* Revision 1.3 1997/12/14 14:37:18 ariels
* Changed "Inconsistent directions" warning to include the names of the
* ests (previously, only the name of the clone was included, which was
* useless for using the output).
*
* Revision 1.2 1997/11/30 07:58:19 ariels
* Added ChangeLog comment and static rcsid string.
*
static char rcsid[] = "$Id: est_table.c,v 1.23 1998/06/29 12:16:14 avner Exp $";
#include <stdio.h>
#include "est_table.h"
#include "error.h"

extern int bound,cutoff,gapext,gapop,match,mismatch; /* global arguments */
extern new_cluster,new_size;
extern char p_ests,p_htops,dirty_tails;

static table_type est_table;

int look_up(char *key)
{
    int i;
    for (i = 0; i < est_table.size; i++)
        if (!strcmp(est_table.arr[i]->key,key)) return i;
    return -1;
}

/* add a node with key <key> to the table (unless already exists) */
void est_table__add(rich_fasta_seq_ptr rfp)
{
    int idx = est_table.size++;
    est_table.arr[idx] = (est *) malloc(sizeof(est));
    if (!est_table.arr[idx])
        err("FATAL ERROR: Can't allocate memory for cluster. aborting\n");
    strcpy(est_table.arr[idx]->key,rfp->name);
    est_table.arr[idx]->seq_data = rfp;
    est_table.arr[idx]->in_degree=0;
    est_table.arr[idx]->invert_algn_color=0;
    est_table.arr[idx]->ordinal=0;
    est_table.arr[idx]->olaps=NULL;
}

```

```

est_table.arr[idx]->clone_pair_idx=1;
est_table.arr[idx]->node_path = NULL;
est_table.arr[idx]->hyper_edge_path = NULL;
est_table.arr[idx]->hyper_edge = NULL;
est_table.arr[idx]->hyper_edge_len = 0;
est_table.arr[idx]->ximeric = 0;
est_table.arr[idx]->test_case = 0;
}

int est_table__len(int idx)
{
    return est_table__original_len(idx);
}

int est_table__original_len(int idx)
{
    return est_table.arr[idx]->seq_data->original_len;
}

int est_table__cleaned_len(int idx)
{
    return est_table__first_dirty(idx) - est_table__first_clean(idx);
}

rich_fasta_seq_ptr est_table__seq_data(int idx) {
    return est_table.arr[idx]->seq_data;
}

int est_table__is_rna(int idx)
{
    return (est_table.arr[idx]->seq_data &&
            est_table.arr[idx]->seq_data->type &&
            strcmp(est_table.arr[idx]->seq_data->type, "RNA")==0);
}

int est_table__is_est(int idx)
{
    return (est_table.arr[idx]->seq_data &&
            est_table.arr[idx]->seq_data->type &&
            strcmp(est_table.arr[idx]->seq_data->type, "EST")==0);
}

int est_table__rna_num() {
    return est_table.rna_num;
}

int est_table__est_num() {
    return est_table.est_num;
}

char *est_table__keywords(int keyword_type, int keyword_id) {
    return est_table.keywords[keyword_type][keyword_id];
}

void est_table__create_keywords() {
    int i,j,k;
    char **temp_keys;
    char *key,*tmp_char;
    int nid;

```

est_table.c

```

temp_keys = (char **) malloc(est_table.size*sizeof(char *));
for(j=0;j<5;j++) {
    for(i=0;i<est_table.size;i++)
        temp_keys[i] = NULL;
    nid=0;
    for (i=0;i<est_table.size;i++) {
        switch (j) {
            case 0:
                key = est_table.arr[i]->seq_data->id?strdup(est_table.arr[i]->seq_data->
                    id):NULL;
                break;
            case 1:
                key = est_table.arr[i]->seq_data->clone?strdup(est_table.arr[i]->seq_data
                    a->clone):NULL;
                break;
            case 2:
                key = est_table.arr[i]->seq_data->tissue?strdup(est_table.arr[i]->seq_da
                    ta->tissue):NULL;
                break;
            case 3:
                key = est_table.arr[i]->seq_data->library?strdup(est_table.arr[i]->seq_d
                    ata->library):NULL;
                break;
            case 4:
                key = est_table.arr[i]->seq_data->chromosome?strdup(est_table.arr[i]->se
                    q_data->chromosome):NULL;
                break;
            default:
                key = NULL;
        }
        if (!key) {
            continue;
        }
        for (k=0;k<nid;k++) {
            if (strcmp(key, temp_keys[k])==0) break;
        }
        if (k==nid)
            temp_keys[nid++]=key;
        else
            free(key);
    }
    est_table.keywords[j] = (char **) malloc((nid+1)*sizeof(char *));
    for(k=0;k<nid;k++)
        est_table.keywords[j][k] = temp_keys[k];
    est_table.keywords[j][nid]=NULL;
}
est_table.rna_num = est_table.est_num=0;
for(i=0;i<est_table.size;i++) {
    est_table.rna_num+=est_table__is_rna(i);
    est_table.est_num+=est_table__is_est(i);
}
free(temp_keys);

int est_table__size()
{
    return est_table.size;
}

char *id(int idx)

```

est_table.c

A-74

```

    {
        return est_table.arr[idx]-->key;
    }

void est_table__increment_deg(int idx)
{
    est_table.arr[idx]-->in_degree++;
}

int est_table__get_deg(int idx)
{
    return est_table.arr[idx]-->in_degree;
}

int est_table__get_inverse_color(int idx)
{
    return est_table.arr[idx]-->invert_algn_color;
}

void est_table__put_inverse_color(int idx, int clr)
{
    est_table.arr[idx]-->invert_algn_color = clr;
}

int est_table__get_ordinal(int idx)
{
    return est_table.arr[idx]-->ordinal;
}

void est_table__put_ordinal(int idx, int ordinal)
{
    est_table.arr[idx]-->ordinal = ordinal;
}

int est_table__is_inverted(int idx)
{
    return (est_table.arr[idx]-->invert_algn_color==2);
}

char *est_table__original_seq(int idx)
{
    return est_table.arr[idx]-->seq_data-->original_seq;
}

void est_table__put_original_seq(int idx, char *str)
{
    strcpy (est_table.arr[idx]-->seq_data-->original_seq, str);
}

/* char *est_table__cleaned_seq(int idx)
{
    return est_table.arr[idx]-->seq_data-->cleaned_seq;
}
*/

char *est_table__seq(int idx)
{

```

est_table.c

```

    }
    return est_table__original_seq(idx);
}

void est_table__invert_seq(int idx)
{
    char *tmpstr;
    tmpstr = subseq (est_table__original_seq(idx),
                     est_table__original_len(idx)-1, 0);
    est_table__put_original_seq(idx, tmpstr);
    est_table.arr[idx]-->seq_data-->strand = est_table__strand(idx)--1?1:-1;
    free(tmpstr);
}

void est_table__clean() {
    int i, j;
    olap_edge *tmp1, *tmp2;
    for (i=0; i<est_table.size; i++) {
        if (est_table.arr[i]-->seq_data)
            rich_fasta_seq_free(est_table.arr[i]-->seq_data);
        for (tmp1 = est_table__neighbor_list(i); tmp1; ) {
            tmp2 = tmp1-->next;
            free(tmp1);
            tmp1 = tmp2;
        }
        if (est_table.arr[i]-->node_path)
            free(est_table.arr[i]-->node_path);
        if (est_table.arr[i]-->hyper_edge_path)
            free(est_table.arr[i]-->hyper_edge_path);
        if (est_table.arr[i]-->hyper_edge)
            free(est_table.arr[i]-->hyper_edge);
        free(est_table.arr[i]);
    }
    for (i=0; i<5; i++) {
        if (!est_table.keywords[i]) continue;
        j=0;
        while (est_table.keywords[i][j]) {
            free(est_table.keywords[i][j]);
            j++;
        }
        free(est_table.keywords[i]);
        est_table.keywords[i] = NULL;
    }
    est_table.size = 0;
}

int get_cluster_volume(void)
{
    int i, volume=0;
    for (i=0; i<est_table.size; i++)
        volume += est_table__len(i);
    return volume;
}

```

est_table.c

A-7S

```

void est_table__print(FILE *fp) {
    int i;
    for (i=0; i<est_table__size(); i++)
        rich_fasta_write(fp, est_table.arr[i]-->seq_data);
}

void est__print(FILE *fp, int est_idx, int ordinal)
{
    fprintf(fp, ">%s\t#IDX %d\t#LN %d\t(%d'th est)\n",
            id(est_idx), est_idx, est_table__len(est_idx), ordinal);
    if (p_ests=="y")
        write_sequence(fp, est_table__seq(est_idx));
}

void est_table__add_edge(int idx1, int idx2, int st1, int st2, int endl, int end2)
{
    int new_edge=0;
    olap_edge *tmp = est_table__neighbors(idx1, idx2);
    if (tmp)
    {
        if (abs(tmp->endl-tmp->st1) > abs(endl-st1)) return;
    }
    else
    {
        new_edge=1;
        tmp = (olap_edge *) malloc(sizeof(olap_edge));
        if (!tmp)
            err("ERROR: can't allocate memory for olap edge\n");
    }
    tmp->est_id = idx2;
    tmp->st1 = st1;
    tmp->st2 = st2;
    tmp->endl = endl;
    tmp->end2 = end2;
    tmp->inverted = (st2 > end2 || st1 > endl);
    if (new_edge)
    {
        tmp->next = est_table.arr[idx1]-->olaps;
        est_table.arr[idx1]-->olaps = tmp;
    }
    if (new_edge)
        tmp = (olap_edge *) malloc(sizeof(olap_edge));
    else
        tmp = est_table__neighbors(idx2, idx1);
    if (!tmp)
        err("ERROR: can't allocate memory for olap edge\n");
    tmp->est_id = idx1;
    tmp->st1 = st1;
    tmp->st2 = st2;
    tmp->endl = endl;
    tmp->end2 = end2;
    tmp->inverted = (st2 > end2 || st1 > endl);
    if (new_edge)
    {
        tmp->next = est_table.arr[idx2]-->olaps;
        est_table.arr[idx2]-->olaps = tmp;
    }
}

```

est_table.c

```

olap_edge *est_table__neighbors(int idx1, int idx2) {
    olap_edge *tmp;
    for (tmp = est_table.arr[idx1]-->olaps; tmp && (tmp->est_id != idx2); tmp = tmp->next)
        return tmp;
}

olap_edge *est_table__neighbor_list(int idx)
{
    return est_table.arr[idx]-->olaps;
}

void est_table__put_variant(int est_idx, int variant)
{
    est_table.arr[est_idx]-->variant_no = variant;
}

int est_table__get_variant(int est_idx)
{
    return est_table.arr[est_idx]-->variant_no;
}

/* the DR field data */
int est_table__direction(int idx)
{
    return est_table.arr[idx]-->seq_data->direction;
}

/*
 * did we inverted this sequence (namely, did we predict it is from the 3'
 * strand) ? 1 for not inverted, -1 for inverted
 */
int est_table__strand(int idx)
{
    return est_table.arr[idx]-->seq_data->strand;
}

int est_table__inversion_score(int i)
{
    if (est_table__direction(i)==5)
        if (est_table__get_inverse_color(i)==1) return 1;
    else return -1;
    if (est_table__direction(i)==3)
        if (est_table__get_inverse_color(i)==1) return -1;
    else return 1;
    return 0;
}

int est_table__clone_mate(int i)
{
    return est_table.arr[i]-->clone_pair_idx;
}

void est_table__pair_clones(void)
{
    int i, j;
    int other_direction; /* Direction we're looking for */
}

```

est_table.c

```

for(i=0; i<est_table__size(); i++)
    if (est_table.arr[i]->seq_data->clone &&
        est_table__direction(i)) {
        other_direction = 8 - est_table__direction(i);
        for(j=i+1; j<est_table__size(); j++)
            if (est_table.arr[j]->seq_data->clone &&
                strcmp(est_table.arr[i]->seq_data->clone,
                    est_table.arr[j]->seq_data->clone) == 0)
                if (est_table__direction(j) == other_direction) {
                    est_table.arr[i]->clone_pair_idx = j;
                    est_table.arr[j]->clone_pair_idx = i;
                }
            else
                record_warn("Inconsistent directions (%d, %d) for clone %s "
                    "ests %s, %s\n",
                    est_table__direction(i), est_table__direction(j),
                    est_table.arr[i]->seq_data->clone,
                    id(i), id(j));
        }
    }

void est_table__set_node_path(int est, int *path, int num)
{
    int i;
    if ((est_table.arr[est]->node_path == (int *)
        malloc(num * sizeof(int))) == NULL)
        err("memory failure for est node_path (%d)", num);
    for(i=0; i<num; i++)
        est_table.arr[est]->node_path[i] = path[i];

    int est_table__node_path_pos(int est, int pos)
    {
        return est_table.arr[est]->node_path[pos];
    }

    char* est_table__get_clone(int est)
    {
        return est_table.arr[est]->seq_data->clone;
    }

    int est_table__first_clean(int idx)
    {
        return est_table.arr[idx]->seq_data->first_clean;
    }

    int est_table__first_dirty(int idx)
    {
        return est_table.arr[idx]->seq_data->first_dirty;
    }

    int est_table__trim5(int idx)
    {
        return est_table.arr[idx]->seq_data->trim5;
    }

```

```

int est_table__trim3(int idx)
{
    return est_table.arr[idx]->seq_data->trim3;
}

int est_table__any_ximeric()
{
    int i;
    for (i=0; i<est_table__size(); i++)
        if (est_table.arr[i]->ximeric) return 1;
    return 0;
}

void est_table__set_ximeric(int idx)
{
    est_table.arr[idx]->ximeric=1;
}

int est_table__is_ximeric(int idx)
{
    return est_table.arr[idx]->ximeric;
}

void est_table__set_test_case(int idx)
{
    est_table.arr[idx]->test_case = 1;
}

int est_table__test_case(int idx)
{
    return est_table.arr[idx]->test_case;
}

int est_table__contig(int idx)
{
    return est_table.arr[idx]->contig;
}

void est_table__set_contig(int idx, int contig)
{
    est_table.arr[idx]->contig = contig;
}

void est_table__mark_alignment_range(int idx, int start, int end)
{
    est_table.arr[idx]->start_in_cons = start;
    est_table.arr[idx]->end_in_cons = end;
}

int est_table__get_start_in_cons(int idx)
{
    return est_table.arr[idx]->start_in_cons;
}

int est_table__get_end_in_cons(int idx)
{
    return est_table.arr[idx]->end_in_cons;
}

```



```

char *est_table__clone_name(int idx)
{
    return est_table.arr[idx]->seq_data->clone;
}

int est_table__clone_length(int five_p, int three_p)
{
    int len;

    if ((len = est_table.arr[five_p]->seq_data->clone_len) !=
        est_table.arr[three_p]->seq_data->clone_len) {
        record_warn("clone length field for clone mates"
            "%s and %s [clone %s] is different",
            id(five_p), id(three_p), est_table__clone_name(five_p));
        return 0;
    }
    return len;
}

int *est_table__get_node_path(int est)
{
    return est_table.arr[est]->node_path;
}

int *est_table__get_hyper_edge_path(int idx)
{
    return est_table.arr[idx]->hyper_edge_path;
}

void est_table__set_hyper_edge_path(int idx, int *hyperedge)
{
    est_table.arr[idx]->hyper_edge_path = hyperedge;
}

hyper_edge_node *est_table__get_hyper_edge(int idx)
{
    return est_table.arr[idx]->hyper_edge;
}

void est_table__set_hyper_edge(int idx, hyper_edge_node *hyperedge)
{
    est_table.arr[idx]->hyper_edge = hyperedge;
}

int est_table__get_hyper_edge_len(int idx)
{
    return est_table.arr[idx]->hyper_edge_len;
}

void est_table__set_hyper_edge_len(int idx, int len)
{
    est_table.arr[idx]->hyper_edge_len = len;
}

/*
 * est_table_switch_node_path_hyper_edge -
 * switch the pointers of the two arrays, should be remove when we
 * start using only one of them.
 */
void est_table_switch_node_path_hyper_edge()

```

est_table.c

```

int i;
int *tmp;
for(i=0; i<est_table__size(); i++) {
    tmp = est_table.arr[i]->hyper_edge_path;
    est_table.arr[i]->hyper_edge_path=est_table.arr[i]->node_path;
    est_table.arr[i]->node_path=tmp;
}

void est_table__cut_head(int est_idx, int cut_len)
{
    rich_fasta_seq__trim_head(est_table.arr[est_idx]->seq_data, cut_len);
}

/*
 * note - a differnet conention as to the parameter of cuts from the end
 * exists between this function, and the lower 'est_table__cut_tail'.
 * we pay for it here.
 */
void est_table__cut_tail(int est_idx, int cut_len)
{
    rich_fasta_seq__trim_tail(est_table.arr[est_idx]->seq_data,
        est_table__len(est_idx) - cut_len);
}

void est_table__set_component(int idx, int component)
{
    est_table.arr[idx]->component = component;
}

int est_table__get_component(int idx)
{
    return est_table.arr[idx]->component;
}

int est_table__verify_est_path(int idx)
{
    int i, rc=1;
    for(i=0;
        est_table.arr[idx]->node_path[i] != -1 &&
        est_table.arr[idx]->hyper_edge_path[i] != -1
        ; i++) {
        if(est_table.arr[idx]->node_path[i] !=
            est_table.arr[idx]->hyper_edge_path[i]) {
            rc=0;
            break;
        }
    }
    if(est_table.arr[idx]->node_path[i] != -1 ||
        est_table.arr[idx]->hyper_edge_path[i] != -1)
        rc=0;
    if(rc==0) {
        printf("test node and hyper_edge disagree on est %d path\n", idx);
        printf("hyper_edge:");
        for(i=0; est_table.arr[idx]->hyper_edge_path[i] != -1; i++)
            printf(" %d", est_table.arr[idx]->hyper_edge_path[i]);
        printf("\n");
        printf("test node:");
        for(i=0; est_table.arr[idx]->node_path[i] != -1; i++)
            printf(" %d", est_table.arr[idx]->node_path[i]);
    }
}

```

est_table.c

```
    )
    printf("\n");
    return rc;
}

int est_table__next_in_variant(int id, int start_seq)
{
    int seq_idx = start_seq;
    while ( seq_idx < est_table__size())
        if (est_table__get_variant(seq_idx) == id)
            return seq_idx;
    return -1;
}
```



```

/* Flipper -- cluster-based contig flipping
*/
*/
*/
*/
$Log: flipper.c,v $
Revision 1.8 1998/06/18 14:42:47 ariels
Strip "ifdef CPROF_ALIGN" and "ifdef IMPROVE_CPROF" lines.
Revision 1.7 1998/04/24 12:49:13 avner
'rich_fasta_read_seq' is called with the FLIPPER_MODE parameter.
Revision 1.6 1998/04/24 09:31:59 avner
add printing of cmd-line.
change parameters. no more defaults for files to save errors.
gb_ver is set to 905.
Revision 1.5 1998/03/29 20:52:10 avner
use a call to 'est_table__original_seq' instead the previous use of wrong
field (due to uncleaning).
Revision 1.4 1998/02/22 17:23:41 ariels
Kluge rich-Fasta sequence output as follows, to ensure all fields are
copied from input to output: Print the original sequence header,
tacking on the appropriate 'ST' field. This relies on not having a
'ST' field on the input, and on not changing any other fields.
This kluge will be eliminated when we re-write the rich-fasta parser.
Revision 1.3 1998/02/20 12:48:25 avner
change semantics of returned value in 'read_cluster_sequences'. -1 is
a failure, while 0 is a cluster that can't be fully read (bigger then
MAX_CLUSTER_SIZE). Previously it was trimmed, and this is bad, since he
will later be pissed-off with having 'unexisting' sequences in the pairs file
Revision 1.2 1998/02/17 16:14:43 ariels
Do 'something' with large (> MAX_CONTIGS contigs) clusters.
Improve source layout.
Revision 1.1 1998/02/11 16:23:18 ariels
Initial revision
*/

static char rcsid[] = "$Id: flipper.c,v 1.8 1998/06/18 14:42:47 ariels Exp $";

#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <string.h>
#include <float.h>
#include <assert.h>

#include "prm.h"
#include "error.h"
#include "rf.h"
#include "general.h"
#include "olap_graph.h"

```

```

#include "est_table.h"

/* Most contigs to handle in a cluster. Running time is exponential in this! */
/* Also, an "int" should contain > MAX_CONTIGS bits.... */
#define MAX_CONTIGS 24
#define REAL_MAX_CONTIGS 2000

/* General quadratic form */
/* Defines v --> v^t A v + <b, v> + c */
/* Store only upper-triangular part of A */
typedef struct quadratic_forms {
    float a[MAX_CONTIGS*MAX_CONTIGS];
    float b[MAX_CONTIGS];
    float c;
} quadratic_form;

/* globals */
FILE *pairs_file, *desired_file;
int gb_ver;

typedef char cluster_name_t[CLUSTER_NAME_SIZE];
cluster_name_t cluster_id, first, last;
int min_size, max_size;
int n_contigs;

int match, mismatch, gapop, gapext, lgapop, lgapext;
int cutoff, bound;
int color_algn_len;
char p_pairs, dirty_tails = 'y';

/* Unique flipper penalties */
float reverse_rna; /* absolute */
float reverse_est; /* absolute */
float break_clone_pair; /* relative */

char merge_contigs;
char write_pairs_file;
char p_est = 'y';

void prologue(int ac, char *av[],
               fasta_file *seqfile, FILE **pairs_file, FILE **desired_file,
               FILE **out_file)
{
    char dir[UNIX_FNAME_LEN], fname[UNIX_FNAME_LEN], name[UNIX_FNAME_LEN],
    desired[UNIX_FNAME_LEN], pairs[UNIX_FNAME_LEN], out[UNIX_FNAME_LEN];
    struct stat st_buf;
    int i;

    /* Quick re-alignment instructions:
    M-x load-file RET -ariels/align-regexp.elc RET
    Set the region to the arguments
    M-x align-regexp RET = RET
    M-x align-regexp RET & RET
    M-x align-regexp RET ! RET
    */
}

```

```

*/
for (i=0; i<ac; i++)
    printf("%s ", av[i]);
printf("\n");
fflush(stdout);

prm_argv
(ac, av, P_PRINT | P_HELP /* broken prm_argv ! | P_ERROR */ ,

/* Files */
"dir" =
"in" =
frame,
"pairs_file" =
on file",
pairs,
"write_pairs_file" = n
&write_pairs_file,
"out" =
", out,

/* Filters */
"desired_file" = none
contigs",
desired,
"first" =
o analyze",
first,
last,
analyze",
last,
"min_size" = 1
size to analyze",
&min_size,
"max_size" = 50000
size to analyze",
&max_size,

/* six18 parameters */
"match" = 4
match",
"mismatch" = -9
mismatch",
&mismatch,
"gapop" = 8
gapop",
&gapop,
"gapext" = 4
gapext",
&gapext,
"lgapop" = 48
lgapop",
&lgapop,
"lgapext" = 0
lgapext",
&lgapext,

```

```

"bound" = 50
or successful alignment",
&bound,
"cutoff" = 40
efines hills)",
&cutoff,

/* Flipper particularities */
"color_algn_len" = 35
o color overlap graph",
&color_algn_len,
"reverse_rna" = 3.0
rse an rna strand",
&reverse_rna,
"reverse_est" = 1.0
rse an est strand",
&reverse_est,
"break_clone_pair" = 2.0
k clone data",
&break_clone_pair,

"merge_contigs" = n
gs in cluster (N/Y)",
&merge_contigs,

/* Debugging information, mostly */
"p_pairs" = n
o (y/n)",
&p_pairs,
"gb_pairs" = 905
ersion",
&gb_ver,
EOLIST);

init_matrix(match, mismatch, gapext, gapop);
init_long_gaps(lgapext, lgapop);

*desired_file = "pairs_file" = NULL;
sprintf(name, "%s/%s", dir, fname);

if ((*seqfile = rich_fasta_open(name)) == NULL)
    err("can't open sequence file %s", name);
switch(write_pairs_file) {
case 'n': case 'r':
    if (pairs[0] == '\0')
        *pairs_file = NULL;
    else
        if ((*pairs_file = fopen(pairs, "r")) == NULL)
            err("couldn't open pairs file %s", pairs);
    break;
case 'y': case 'w': case 'c':
    if (stat(pairs, &st_buf) == 0) /* file exists! */
        if (write_pairs_file != 'c')
            err("%s exists! (use write_pairs_file=c)", pairs);
    else
        fprintf(stderr, "overwriting pairs file %s", pairs);
    if ((*pairs_file = fopen(pairs, "w")) == NULL)
        err("couldn't open %s for write", pairs);
    break;
}

```

```

case 'a':
    if ((*pairs_file = fopen(pairs, "a")) == NULL)
        err("couldn't open %s for append", pairs);
    break;
default:
    err("write_pairs_file must be one of (n|o|y|es|c|reate|)");
    break;
}

if (strcmp(desired, "none") != 0)
    if ((*desired_file = fopen(desired, "r")) == NULL)
        err("couldn't open desired file %s\n", desired);

if ((*out_file = fopen(out, "a")) == NULL)
    err("couldn't open output file %s for append", out);
}

```

```

static int get_next_desired(FILE *f, char *desired)
{
    char *p;

```

```

    if (fscanf(f, "%s", desired) == 0)
        return 0;
    if ((p = strchr(desired, '.')) != NULL)
        *p = '\0';
    /* strip CLUSTER.CONTIG */
}

```

```

/*
 * Compare two ontig identifiers, in CLUSTER[.CONTIG] format. If
 * .CONTIG is empty, CLUSTER will match *any* contig identifier with
 * the same cluster name.

```

```

 * Return <0, 0, >0 to indicate first argument comes before, together
 * with or after the second argument.
 */

```

```

int cid_cmp(char *c1, char *c2)
{
    char *d1 = strdup(c1), *d2 = strdup(c2);
    char *e1, *e2;
    int res;

    if (e1 = strchr(d1, '.'))
        *e1++ = '\0';
    if (e2 = strchr(d2, '.'))
        *e2++ = '\0';

    if (! (res = strcmp(d1, d2))) && e1 && e2
        res = strcmp(e1, e2);
    free(d1);
    free(d2);
    return res;
}

```

```

void print_title(void)
{
    printf("#####\n");
    printf("### begin cluster %s-15s (%4d contigs, %5d ests) ###\n",

```

```

    cluster_id, n_contigs, est_table_size());
    printf("#####\n");
}

void report_cluster(void)
{
    char *msgs[4] = {"good", "warnings", "errors", "warnings and errors"};

    if (errd()) {
        report_err(stdout, "error: ");
        clear_err();
    }
    if (warned()) {
        report_warn(stdout, "warning: ");
        clear_warn();
    }

    /* !ix == (x ? 1 : 0), for "obvious" reasons */
    printf("Report for cluster %s: %s\n",
        cluster_id, msgs[2*!errd() + !warned()]);

    printf("\n\n");
}

```

```

/* Calculate penalty for keeping contig #CTG_NUM in current orientation */
float calc_contig_penalty(int ctg_num)
{
    float score = 0.0;
    int i, j;

```

```

    for(i=0; i<est_table_size(); i++)
        if (est_table_contig(i) == ctg_num) {
            short single = est_table__is_rna(i) ||
                (est_table__clone_mate(i) == -1);
            short threep = est_table__direction(i) == 3;

            score += (est_table__is_inverted(i) ? -1 : 1) *
                (est_table__is_rna(i) ? reverse_rna : reverse_est) *
                (single ? 1.0 : 0.5) *
                (threep ? -1.0 : 1.0);
        }
}

```

```

for(i=0; i<est_table_size(); i++)
    if ((est_table_contig(i) == ctg_num) &&
        (j = est_table__clone_mate(i) != -1) &&
        (est_table_contig(j) == ctg_num) &&
        est_table__direction(i) && est_table__direction(j)) {
        float incr = est_table__direction(i) != est_table__direction(j) ?
            -break_clone_pair : break_clone_pair;
        score += incr;
    }

    return score;
}

```

```

quadratic_form calc_penalties(int num)

```

```

/* quadratic_form Q;
int i, j;

/* zero */
Q.c = 0.0;
for(i=0; i<num; i++) {
    Q.b[i] = 0.0;
    for(j=0; j<num; j++)
        Q.A[i*num+j] = 0.0;
}

/* single-est scores */
for(i=0; i<est_table_size(); i++) {
    short single = est_table_is_rna(i) || (est_table_clone_mate(i) == -1);
    short threep = est_table_direction(i) == 3;
    Q.b[est_table_contig(i)] +=
        (est_table_is_inverted(i) ? -1 : 1) *
        (single ? 1.0 : 0.5) *
        (threep ? -1.0 : 1.0);
}

/* combined scores */
for(i=0; i<est_table_size(); i++)
    if ((j = est_table_clone_mate(i)) != -1 &&
        est_table_direction(i) && est_table_direction(j)) {
        int contig_i = est_table_contig(i);
        int contig_j = est_table_contig(j);
        float incr = est_table_direction(i) != est_table_direction(j) ?
            -break_clone_pair : break_clone_pair;
        if (est_table_is_inverted(i)) incr = -incr;
        if (est_table_is_inverted(j)) incr = -incr;
        if (contig_i == contig_j)
            Q.c += incr;
        else
            Q.A[contig_i*num+contig_j] += incr;
    }

printf("Q.A:\n");
for(i=0; i<num; i++) {
    for(j=0; j<num; j++)
        printf("%f ", Q.A[i*num+j]);
    putchar('\n');
}

printf("Q.b:\n");
for(i=0; i<num; i++)
    printf("%f ", Q.b[i]);
printf("\nQ.c = %f\n", Q.c);

return Q;
}

void report_problems(float v[MAX_CONTIGS], int num)
{
    int i, j;

```

flipper.c

```

/* single-est problems */
for(i=0; i<est_table_size(); i++) {
    short threep = est_table_direction(i) == 3;
    short invert = est_table_is_inverted(i);
    if (threep) invert = !invert;
    if (v[est_table_contig(i)] < 0) invert = !invert;
    if (invert)
        record_warn("Poor inversion of %s", id(i));
}

for(i=0; i<est_table_size(); i++)
    if ((j = est_table_clone_mate(i)) != -1 &&
        est_table_direction(i) && est_table_direction(j)) {
        int contig_i = est_table_contig(i);
        int contig_j = est_table_contig(j);
        short invert = est_table_direction(i) != est_table_direction(j);
        if (est_table_is_inverted(i)) invert = !invert;
        if (est_table_is_inverted(j)) invert = !invert;
        if (v[est_table_contig(i)] * v[est_table_contig(j)] < 0)
            invert = !invert;
        if (invert)
            record_warn("Poor inversion of the pair <%s,%s>", id(i), id(j));
    }
}

float eval_quadratic(quadratic_form Q, float v[MAX_CONTIGS], int num)
{
    int i, j;
    float score = Q.c;

    for(i=0; i<num; i++)
        score += Q.b[i] * v[i];
    for(j=0; j<num; j++)
        for(i=0; i<num; i++)
            score += Q.A[i*num+j] * v[i] * v[j];
    return score;
}

int read_cluster_sequences(fasta_file ff, char *cluster_id)
{
    static rich_fasta_seq_ptr rfp = NULL;
    static int first = 1;
    int status = 1;
    char cur_contig[CLUSTER_NAME_SIZE];

    cluster_id[0] = '\0';
    n_contigs = 1;

    if (first) {
        rfp = rich_fasta_read_seq(ff, FLIPPER_MODE);
        first = 0;
    }
    if (!rfp) return -1;
    strcpy(cluster_id, rfp->meta_cluster);
    strcpy(cur_contig, rfp->cluster);
    while (rfp && strcmp(cluster_id, rfp->meta_cluster) == 0) {
        if (strcmp(cur_contig, rfp->cluster) != 0) {

```

flipper.c


```

++n_contigs;
strcpy(cur_contig, rfp->cluster);
}
if (est_table_size() < MAX_CLUSTER_SIZE) {
    est_table_add(rfp);
    est_table_set_contig(est_table_size()-1, n_contigs-1);
} else {
    record_err("est table full; %s ignored", rfp->id);
    status = 0;
}
rfp = rich_fasta_read_seq(ff, FLIPPER_MODE);
}
return status;
}
/* code originally from olap_graph */

```

```

/*=====
/*
* the next 2 functions are used for scanning the alignments-graph, and
* 2-colors it in such a way that regular alignments are monochromatic,
* and inverted ones are not. Note that in the usual case (running on a
* clustering output) we get a tree of alignment, hence most of what we
* do here is obsolete, but still correct.
*/
int color_inverse(void)
{
    int initial_component_vertex, component_no = 0;
    int inconsistent = 0;
    int color_recurr(int, int);

    for (initial_component_vertex = 0;
         initial_component_vertex < est_table_size();
         initial_component_vertex++) {
        if (est_table_get_inverse_color(initial_component_vertex)) continue;
        component_no++;
        if (color_recurr(initial_component_vertex, 1) == -1)
            inconsistent = 1;
    }
    assert(component_no >= n_contigs); /* internal consistency */
    if (component_no > n_contigs)
        record_warn("cluster is disconnected (%d contigs, %d components)",
                    n_contigs, component_no);
    if (inconsistent) {
        print_inverses();
        record_warn("reverse-alignment-info inconsistent");
        return 0;
    }
    return 1;
}

int color_recurr(int idx, int color)
{
    int neighbor, rc;
    olap_edge *edge;

```

```

if (est_table_get_inverse_color(idx) > 0)
    if (est_table_get_inverse_color(idx) == color)
        return 0; /* compatibility */
    else
        return -1;
est_table_put_inverse_color(idx, color);
for (edge = est_table_neighbor_list(idx); edge; edge = edge->next) {
    if (abs(edge->endl - edge->stl) >= color_algn_len) {
        rc = color_recurr(edge->est_id, edge->inverted?3-color:color);
        if (rc != 0) return rc;
    }
}
return 0;
}

/*=====
/*
* determine order in which fragments will be processed.
* Normally the order is produced by iteratively taking a fragment
* that align the most (to some degree of approximation) with the component
* of the previous fragments/
*/
int order_ests(int *order_arr, int start, int end)
{
    int next_in_order, ordinal;
    int get_max_aligned(int first, int start, int end);

    if (start+1 == end) {
        order_arr[start] = start;
        return 1;
    }

    for (ordinal = start+1; ordinal <= end; ordinal++) {
        next_in_order = get_max_aligned(ordinal==start+1, start, end);
        if (next_in_order >= 0) {
            order_arr[ordinal-1] = next_in_order;
            est_table_put_order(next_in_order, ordinal);
        }
        else
            return 0;
    }
    return 1;
}

/*
* Get the next fragment in the order by looking for the fragment that
* (i) is not ordered yet (ordinal==0).
* (ii) has maximal alignment with some fragment that was already ordered.
* if first != 0 then nothing was ordered yet, and one of the fragments in the
* maximal length alignment is returned.
*/
int get_max_aligned(int first, int start, int end)
{

```



```

if (score > best) {
    memcpy(best_v, v, MAX_CONTIGS * sizeof(float));
    best = score;
}

printf("Cluster %s score = %f\n", cluster_id, best);
else {
    float score = 0, this;
    record_warn("Too many (%d) contigs; flipping each separately",
               n_contigs);
    if (n_contigs > REAL_MAX_CONTIGS)
        record_err("More than %d contigs!",
                   REAL_MAX_CONTIGS);
    for (i=0; i < n_contigs; i++) {
        this = calc_contig_penalty(i);
        printf("Cluster %s, contig %d score = %f\n", cluster_id, i, this);
        best_v[i] = this < 0.0 ? -1.0 : 1.0;
    }
}

/* Actually flip contigs */
for (i=0; i < est_table__size(); i++)
    if (best_v[est_table__contig(i)] < 0)
        est_table__invert_seq(i);

report_problems(best_v, n_contigs);

```

```

/* Output contigs */
j = 0;
k = 0;
for (i=0; i < n_contigs; i++) {
    /* find where this contig ends */
    while (++j < est_table__size()) &&
        est_table__contig(j) == est_table__contig(k))
        ;
    /* order sequences in the contig */
    if (!order_ests(order, k, j))
        record_warn("Disconnected pairs information for contig %s",
                    est_table__seq_data(k)->cluster);
    /* output the contig */
    while (k < j) {
        /* Can't use this...
        rich_fasta_write(out_file, est_table__seq_data(order[k++])));
        because then we don't output all fields.
        Instead, we use this HACKED KLUGE, which won't be here in
        the next version (106). */
        char *p, *q;
        for (p=est_table__seq_data(order[k])->header;
             *p && *p != '\n';
             p++)
            putc(*p, out_file);
        fprintf(out_file, " %s %c\n",
                est_table__seq_data(order[k])->strand >= 0 ? '+' : '-');

```

```

p = est_table__original_seq(order[k]);
q = p + strlen(p);
while (p < q) {
    fprintf(out_file, "%.60s\n", p);
    p += 60;
}
k++;
}

printf("\ninitial order is \n");
print_ests_order(order);

/* *** DBG *** */
for (i=0; i < est_table__size(); i++)
    putchar(est_table__seq_data(i)->strand >= 0 ? '+' : '-');
    putchar('\n');

report_cluster();
while (1);

return 0;
}

```



```
/*
 * Copyright 1996 Compugen, Ltd.
 * Authorization to use this code is given solely to Compugen customers.
 * This authorization is limited by the terms of the Compugen Hardware and
 * Software License Agreement.
 *
 * Last update: $Date: 1997/11/30 07:58:59 $ by $Author: ariels $
 * Revision: $Revision: 1.2 $
 */
```

```
/* $Log: hilltops.c,v $
 * Revision 1.2 1997/11/30 07:58:59 ariels
 * Added Changelog comment and static rcsid string.
 */
```

```
static char rcsid[] = "$Id: hilltops.c,v 1.2 1997/11/30 07:58:59 ariels Rel $";
```

```
#include <stdio.h>
#include <strings.h>
#include <sys/types.h>
#include <malloc.h>
#include <math.h>
```

```
#include "hilltops.h"
```

```
/*=====
char *subseq (char *seq, int first, int last)
/*
 * Cut a sub-sequence from a given input sequence. If first > last, flip it
 * and translate C<->O, A<->T, (and subsets as well).
 */
```

```
{
    char *res;
    int i, len, newlen, flip = 0;

    len = strlen (seq);

    if (first < 0) first = 0;
    if (first >= len) first = len-1;
    if (last < 0) last = 0;
    if (last >= len) last = len-1;

    newlen = last - first;
    if (newlen < 0) {
        flip = 1;
        newlen = -newlen;
    }
    newlen++;
    if (! (res = (char *) malloc (newlen+1)))
        error(1, -1, "subseq: can't allocate string\n");

    res[newlen] = '\0';
    if (flip)
        for (i = 0; i < newlen; i++)
            res[i] = inv(seq[first-i]);
    else
        for (i = 0; i < newlen; i++)
```

hilltops.c

```
    res[i] = seq[first+i];
    return (res);
}
```

```
/*=====
char inv (char c)
/*
 * return base pair
 */
```

```
{
    char alphabet[] = "-ACMGKRSVTWYHKDN";
    /* K = G or T M = A or C R = G or A S = G or C W = A or T Y = T or C
     * B = G or T or C D = G or A or T H = A or C or T V = G or C or A
     * N = A or C or G or T */
    if (c == 'A') return 'T';
    if (c == 'C') return 'G';
    if (c == 'G') return 'C';
    if (c == 'T') return 'A';
    if (c == '-') return '-';
    if (c == 'N') return 'N';
    if (c == 'R') return 'Y';
    if (c == 'Y') return 'R';
    if (c == 'K') return 'M';
    if (c == 'M') return 'K';
    if (c == 'S') return 'S';
    if (c == 'W') return 'W';
    if (c == 'B') return 'V';
    if (c == 'D') return 'H';
    if (c == 'H') return 'D';
    if (c == 'V') return 'B';
    if (c == 'X') return 'X';
    return c;
}
```

```
/*=====
void create_segment (segment **obj)
{
    if (! ((*obj) = (segment *) malloc (sizeof(segment))))
        error(1, -1, "couldn't allocate memory for segment.\n");

    (*obj)->next = NULL;
    (*obj)->start = 0;
    (*obj)->end = 0;
    (*obj)->score = 0.0;
    (*obj)->best_y = 0;
    (*obj)->x0 = 0;
    (*obj)->y0 = 0;
}

/*=====
void destroy_segment (segment **obj)
{
    free (*obj);
    *obj = NULL;
}

/*=====
void destroy_segment_list (segment **obj)
{
    segment *segp1, *segp2;
```

hilltops.c

A-87

```

for (segp1 = *obj; segp1 != NULL; ) {
    segp2 = segp1->next;
    destroy_segment (&segp1);
    segp1 = segp2;
}
*obj = NULL;
}

/*=====*/
void copy_segment (segment from, segment *obj)
{
    obj->next = from->next;
    obj->start = from->start;
    obj->end = from->end;
    obj->score = from->score;
    obj->best_y = from->best_y;
    obj->x0 = from->x0;
    obj->y0 = from->y0;
}

/*=====*/
void create_hilltop (hilltop **obj)
/*
 * Initialize a sequence alignment structure
 */
{
    if (!(*obj) = (hilltop *) malloc (sizeof(hilltop)))
        error(1, -1, "couldn't allocate memory for hilltop.\n");

    (*obj)->next = NULL;
    (*obj)->type = UNDEFINED;
    (*obj)->final = 0;
    (*obj)->len = 0;
    (*obj)->x0 = 0;
    (*obj)->y0 = 0;
    (*obj)->xt = 0;
    (*obj)->yt = 0;
    (*obj)->score = 0.0;
    (*obj)->area = 0;
    (*obj)->id_percent = 0.0;
    (*obj)->sim_percent = 0.0;

    (*obj)->name[0] = '\0';
    (*obj)->boundary = NULL;
    (*obj)->x = NULL;
    (*obj)->y = NULL;
    (*obj)->diff = NULL;

    /*=====*/
void destroy_hilltop (hilltop **obj)
{
    segment *segp, *segp2;

    if ((*obj)->x != NULL) free ((*obj)->x);
    if ((*obj)->y != NULL) free ((*obj)->y);
    if ((*obj)->diff != NULL) free ((*obj)->diff);
    destroy_segment_list (&((*obj)->boundary));
    free (*obj);
}

```

hilltops.c

```

*obj = NULL;
}

/*=====*/
void destroy_hilltop_list (hilltop **obj)
{
    hilltop *htp1, *htp2;

    for (htp1 = *obj; htp1 != NULL; ) {
        htp2 = htp1->next;
        destroy_hilltop (&htp1);
        htp1 = htp2;
    }
    *obj = NULL;
}

/*=====*/
void copy_hilltop (hilltop from, hilltop *obj)
/*
 * Copy a sequence alignment structure
 */
{
    segment *segp, **seggp;

    obj->next = from->next;
    obj->type = from->type;
    obj->final = from->final;
    obj->len = from->len;
    obj->x0 = from->x0;
    obj->y0 = from->y0;
    obj->xt = from->xt;
    obj->yt = from->yt;
    obj->score = from->score;
    obj->area = from->area;
    obj->id_percent = from->id_percent;
    obj->sim_percent = from->sim_percent;
    strcpy (obj->name, from->name);

    if (! (obj->x = (char *) realloc (obj->x, 1*from->len)))
        error(1, -1, "couldn't allocate memory for hilltop.\n");
    strcpy (obj->x, from->x);

    if (! (obj->y = (char *) realloc (obj->y, 1*from->len)))
        error(1, -1, "couldn't allocate memory for hilltop.\n");
    strcpy (obj->y, from->y);

    if (! (obj->diff = (char *) realloc (obj->diff, 1*from->len)))
        error(1, -1, "couldn't allocate memory for hilltop.\n");
    strcpy (obj->diff, from->diff);

    seggp = &(obj->boundary);
    for (segp = from->boundary; segp != NULL; segp = segp->next) {
        create_segment (seggp);
        copy_segment (*seggp, *seggp);
        seggp = &((*seggp)->next);
    }

    /*=====*/
void update_all_hilltops (segment *seg_list, hilltop **ht_list, int x, int w)

```

hilltops.c

```

/* Use the knowledge of the previous column and the segments of the
 * current column to update the area-map of all locations above cutoff.
 * "w" is used for wraparound. If not zero, hilltops wrap around the
 * horizontal edge of the matrix. In this case w should be equal to "leny".
 */

```

```

int a, b, c, d, delete_flag;
segment *segp1, *segp2, *old_boundary, **segpp;
hilltop *htp, *htp2, **htpp;

/* First, bring segments to correct form (separate x0 and y0) */
for (segp1 = seg_list; segp1 != NULL; segp1 = segp1->next) {
    segp1->y0 = (segp1->x0 & 0xffff);
    segp1->x0 >= 16;
    for (; x - segp1->x0 > (1 << 16); segp1->x0 += (1 << 16));
    for (; (segp1->best_y - segp1->y0 > (1 << 16)); segp1->y0 += (1 << 16));
}

/* Go over the hilltops (which are updated to the previous column) */
for (htp = ht_list; htp != NULL; htp = htp->next) {
    /* cut the boundary information out from the hilltop */
    old_boundary = htp->boundary;
    htp->boundary = NULL;
}

```

```

/* In a hilltop, go over the segments it had in the previous column */
for (segp1 = old_boundary; segp1 != NULL; segp1 = segp1->next) {
    a = segp1->start;
    b = segp1->end;
    /* Now, try to find a segment in the new column which overlaps */
    for (segpp = &seg_list; *segpp != NULL; ) {
        c = (*segpp)->start;
        d = (*segpp)->end;
        if ((c <= b+1 && d >= a) || (w && (c == 0) && (b == w-1))) {
            /* Overlap : note the exact condition! */
            update_hilltop (htp, **segpp, x);
            /* Delete the (used) segment from the list */
            segp2 = (*segpp);
            (*segpp) = (*segpp)->next;
            destroy_segment (&segp2);
        } else /* Only if we don't delete - advance */
            segpp = &((*segpp)->next);
    }
}

```

```

/* We now have the new hilltop boundary, unless it should be merged
 * with a preceding hilltop. Check them all */
for (segp1 = old_boundary; segp1 != NULL; segp1 = segp1->next) {
    a = segp1->start;
    b = segp1->end;
    for (htp2 = ht_list; *htpp != htp; ) {
        delete_flag = 0;
        for (segp2 = (*htpp)->boundary; segp2 != NULL; segp2 = segp2->next) {
            c = segp2->start;
            d = segp2->end;
            if (c <= b+1 && d >= a) { /* Overlap : merge the hilltops */
                merge_hilltops (htp, *htpp);
                /* Delete the merged hilltop from the list */
                htp2 = (*htpp);
                (*htpp) = ((*htpp)->next);
            }
        }
    }
}

```

hilltops.c

```

destroy_hilltop (&htp2);
delete_flag = 1;
break;
}
/* Only if we don't delete - advance */
if (!delete_flag) htp2 = &((*htpp)->next);
}
destroy_segment_list (&old_boundary);
}

```

```

/* create new records for segments which are still unassociated */
for (segp1 = seg_list; segp1 != NULL; segp1 = segp1->next) {
    create_hilltop (&htp);
    update_hilltop (htp, *segp1, x);
    htp->next = *ht_list;
    *ht_list = htp;
}
destroy_segment_list (&seg_list);
}

/*=====
void update_hilltop (hilltop *htp, segment seg, int x)
=====*/
/* We have ascertained that seg is associated with rec. We now have to
 * update the data in rec according to seg.
 */

```

```

segment *segp;

htp->area += seg->end - seg->start + 1;
if (htp->score < seg->score) {
    htp->score = seg->score;
    htp->xt = x;
    htp->yt = seg->best_y;
    htp->x0 = seg->x0;
    htp->y0 = seg->y0;
}
create_segment (&segp);
copy_segment (seg, segp);
segp->next = htp->boundary;
htp->boundary = segp;
}

/*=====
void merge_hilltops (hilltop *htp1, hilltop *htp2)
=====*/
/* Merge the two areas into the first one.
 */

```

```

segment *segp;

htp1->area += htp2->area;
if (htp1->score < htp2->score) {
    htp1->score = htp2->score;
    htp1->xt = htp2->xt;
    htp1->yt = htp2->yt;
    htp1->x0 = htp2->x0;
    htp1->y0 = htp2->y0;
}

```

hilltops.c


```
    }
    for (segp = http2->boundary; segp->next != NULL; segp = segp->next);
    segp->next = http1->boundary;
    http1->boundary = http2->boundary;
    http2->boundary = NULL;
}

/*=====*/
```

```

/* $Log: hyper_graph.c,v $
 * Revision 1.14 1998/06/18 14:42:47 artiels
 * Strip "#ifdef CPROF_ALIGN" and "#ifdef IMPROVE_CPROF" lines.
 *
 * Revision 1.13 1998/06/17 08:44:06 eyal
 * Fix bug in check_est_hyper_edge.
 *
 * Revision 1.12 1998/06/15 12:13:20 eyal
 * Add support for hyper_edge_node list for all the ests, and function which che
ck it
 *
 * Revision 1.11 1998/05/10 07:24:17 eyal
 * 1. fix bug in DBG mode
 * 2. Move find_ests_paths to hyper_graph.h
 *
 * Revision 1.10 1998/05/03 08:41:48 eyal
 * 1. Improve the print out in DBG mode
 * 2. Move est_check out of DBG
 */
static char rcsid[] = "$Id: hyper_graph.c,v 1.14 1998/06/18 14:42:47 artiels Exp $";

#include "est_table.h"
#include "hyper_graph.h"
#include <stdlib.h>
#include <assert.h>

static hyper_graph *hyper_graph_new();
static hyper_edge *hyper_edge_new();
static void hyper_edge_free(hyper_edge *edge);
static int find_hyper_edge_first_node(splice_graph *graph, identifier_t est_id, in
t *min_pos);
static int hyper_edge_passes(int source_idx, int target_idx,
    identifier_t est_id, int pos, splice_graph *graph);
static void hyper_graph_remove_edge(hyper_edge *del_edge, hyper_graph *graph);
static void hyper_edge_print(hyper_edge *edge);
static int hyper_edge_check(hyper_edge *edge);
static void dual_edge_init(dual_edge *de, hyper_edge *source, hyper_edge *target
, int rel);
static void dual_edge_copy(dual_edge *target, dual_edge *source);
static void find_edge_relationship(hyper_graph *graph);
static void create_sub_edges(hyper_graph *graph);

hyper_graph *build_hyper_graph(splice_graph *s_graph) {
    int i, pos, node_idx, nbr;
    identifier_t est_id;
    hyper_edge *edge;
    splice_node *node;
#ifdef DBG
    int j, k, loc, in, end, start;
    cprof p;
#endif
    hyper_graph *h_graph = hyper_graph_new();

```

hyper_graph.c

```

h_graph->s_graph = s_graph;

#ifdef DBG
    for(i=0; i<s_graph->size; i++) {
        pss_graph->node_list[i]->profile;
        for(j=0; j<p->width; j++) {
            in=0;
            for(k=0, loc=p->pos[j]-1; k<p->len; k++) {
                if (!CPROF_IS_BLANK(*cprof_seq_ptr(p, k, j))) {
                    if (!in) {
                        start=k;
                    }
                    in=1;
                    if (CPROF_LETTER_NO(*cprof_seq_ptr(p, k, j)) > 0)
                        loc++;
                    end=k;
                }
                if (in && CPROF_IS_BLANK(*cprof_seq_ptr(p, k, j))) {
                    in=0;
                    break;
                }
            }
            printf("%d\t%d\t%d\t%d\t%d\n", p->id[j], p->pos[j], loc, start, end, i);
        }
    }
#endif

    for (i=0; i<est_table_size(); i++) {
        est_id = est_table_get_id(i); /*
        est_id = i;
        edge = hyper_edge_new();
        edge->est = i;
        edge->graph = s_graph;
        hyper_graph_add_edge(edge, h_graph);
        edge->nodes = est_table_get_hyper_edge_path(est_id);
        edge->len = est_table_get_hyper_edge_len(est_id);
        hyper_edge_check(edge);

        create_sub_edges(h_graph);
        find_edge_relationship(h_graph);
        remove_contained_edges(h_graph);

        return h_graph;
    }

    void check_est_hyper_edge(int est_idx, hyper_edge_node *path, int path_len,
    splice_graph *graph) {
        int i, delta;
        for(i=0; i<path_len; i++) {
            if(i==0) {
                if(path[i].est_start != 0)
                    printf("hyper_edge warning: est %d miss %d bases from the head\n",
                        est_idx, path[i].est_start);
            }
            else {
                if(path[i].cprof_start != 0 &&
                    !path[i].node_id == path[i-1].node_id &&
                    path[i-1].cprof_end < path[i].cprof_start))

```

hyper_graph.c

A-91

```

printf("hyper_edge warning: est %d in node %d start from %d\n",
      est_idx, path[i].node_id, path[i].cprof_start);
delta = path[i].est_start - path[i-1].est_end - 1;
if (delta > 0)
    printf("hyper_edge warning: est %d miss %d bases between %d and %d\n",
          est_idx, delta, path[i-1].node_id, path[i].node_id);
if (path[i].node_id == path[i-1].node_id &&
    (delta = path[i].cprof_start - path[i-1].cprof_end > 0))
    printf("hyper_edge warning: est %d in node %d skip %d bases\n",
          est_idx, path[i].node_id, delta);
}
if (i < path_len-1) {
    delta = splice_node_len(graph->node_list[path[i].node_id])-1 -
    path[i].cprof_end;
    if (delta > 0 &&
        ! (path[i].node_id == path[i+1].node_id &&
            path[i].cprof_end < path[i+1].cprof_start))
        printf("hyper_edge warning: est %d in node %d ends %d from the end\n",
              est_idx, path[i].node_id, delta);
    }
    else {
        /*last one */
        delta = est_table_len(est_idx)-1 - path[i].est_end;
        if (delta > 0)
            printf("hyper_edge warning: est %d miss %d bases from the tail\n",
                  est_idx, delta);
    }
}

void find_ests_paths(splice_graph *graph)
{
    int i, j, rc, pos, min_node_pos, min_node_idx, node_pos, path_len, he_len;
    int path_size;
    identifier_t est_id;
    int *path_node_idx;
    cprof_p;
    splice_node *node;
    hyper_edge_node *he_nodes, *he_node;

    for (j=0; j<est_table_size(); j++) {
        path_size = 100;
        path_len = 0;
        he_len = 0;
        if ((path = (int*) malloc(path_size*sizeof(int))) == NULL)
            err("Memory failure in find_ests_paths for %d ids",
                path_size*sizeof(int));
        path_size = (hyper_edge_node*)
            malloc(path_size*sizeof(hyper_edge_node)) == NULL)
            err("Memory failure in find_ests_paths for %d ids",
                path_size*sizeof(hyper_edge_node));

        /* est_id = est_table_get_id(i); */
        est_id = j;
        node_idx = find_hyper_edge_first_node(graph, est_id, &pos);
        if (node_idx == -1) {
            /* The est does not exist in the graph.
               The only case this can happen is that the est align with no
               other est, and is all dirty ?!
            */

```

```

probably caused by a bug in the uncleaning process or somewhere else.
ones the bug is fixed we can call err() instead of record_warn().
record_warn("Error in find_ests_paths: no first node for est %d\n", est_id);

path[path_len] = -1;
he_nodes[he_len].node_id = -1;
est_table_set_hyper_edge(path(est_id, path));
est_table_set_hyper_edge(est_id, he_nodes);
est_table_set_hyper_edge_len(est_id, path_len);
printf("est %d has no path\n", est_id);
continue;
}
he_node = &he_nodes[0];
p = graph->node_list[node_idx]->profile;

rc = cprof_get_est_start_end(p, pos-1, est_id,
                             &he_node->est_start, &he_node->est_end);
assert(rc);
rc = cprof_get_cprof_start_end(p, pos-1, est_id,
                              &he_node->cprof_start, &he_node->cprof_end);
assert(rc);

while(1) {
    min_node_pos = 10000;
    min_node_idx = -1;
    path[path_len++] = node_idx;
    he_nodes[he_len++].node_id = node_idx;

    node = graph->node_list[node_idx];
    for (i=0; i < graph->size; i++) {
        node_pos = cprof_get_seq_first_location(graph->node_list[i]->profile,
                                                est_id, pos);
        if (node_pos != -1 && min_node_pos > node_pos) {
            min_node_pos = node_pos;
            min_node_idx = i;
        }
        if (min_node_idx == -1) {
            /* we've reached the end */
            break;
        }
        he_node = &he_nodes[he_len];
        p = graph->node_list[min_node_idx]->profile;
        rc = cprof_get_est_start_end(p, min_node_pos-1, est_id,
                                    &he_node->est_start, &he_node->est_end);
        assert(rc);
        rc = cprof_get_cprof_start_end(p, min_node_pos-1, est_id,
                                      &he_node->cprof_start, &he_node->cprof_end);
        assert(rc);

        if (min_node_idx == node_idx) {
            /* we should check if this is a self loop
               or there is a simple cut in the est.
               The real solution is to unite two sequences which
               look like:
               id seq
               -1 AACCTGATC
               1 AACCT
            */

```

```

1      GNTC
2
3      if (he_nodes[he_len-1].cprof_end < he_nodes[he_len].cprof_start) {
4          /* This is not a self edge, so we don't want to add node i to the path
5          again. */
6          if (he_nodes[he_len-1].cprof_end+1 !=
7              he_nodes[he_len].cprof_start) {
8              printf("problem in hyper_edge of %d in node %d\n", est_id, node_idx);
9              cprof_dump(p);
10             }
11             path_len--;
12         }
13         pos = min_node_pos;
14         node_idx = min_node_idx;
15
16         if (he_len == path_size-2) { /* We should allocate more space */
17             if (path = (int*)realloc(path, 2*path_size*sizeof(int))) == NULL)
18                 err("Memory failure in find_est_paths for %d ids",
19                     2*path_size*sizeof(int));
20             if ((he_nodes = (hyper_edge_node*)
21                 realloc(he_nodes, 2*path_size*sizeof(hyper_edge_node))) == NULL)
22                 err("Memory failure in find_est_paths for %d ids",
23                     2*path_size*sizeof(hyper_edge_node));
24             path_size = 2*path_size;
25         }
26         path[path_len] = -1;
27         he_nodes[he_len].node_id = -1;
28         est_table__set_hyper_edge_path(est_id, path);
29         est_table__set_hyper_edge(est_id, he_nodes);
30         est_table__set_hyper_edge_len(est_id, path_len);
31
32         printf("est %d's path: ", est_id);
33         for (i=0; i<path_len; i++)
34             printf("%d ", path[i]);
35         printf("\n");
36         check_est_hyper_edge(est_id, he_nodes, he_len, graph);
37     }
38 }
39
40 hyper_graph *hyper_graph_new() {
41     hyper_graph *new_graph;
42     if (new_graph = (hyper_graph *)malloc(sizeof(hyper_graph))) == NULL)
43         err("memory problem in hyper_graph_new");
44     new_graph->size = 0;
45     return new_graph;
46 }
47
48 void hyper_graph__free(hyper_graph *graph)
49 {
50     int i;
51     for (i=0; i<graph->size; i++)
52         hyper_edge__free(hyper_graph__get_edge(i, graph));
53     free(graph);
54 }
55
56 hyper_edge *hyper_edge_new() {

```

hyper_graph.c

```

hyper_edge * new_edge;
if (new_edge = (hyper_edge *)malloc(sizeof(hyper_edge))) == NULL)
    err("memory problem in hyper_edge_new");
new_edge->out_degree = 0;
new_edge->contained_num = 0;
new_edge->len = 0;
new_edge->is_maximal = 0;
return new_edge;
}

void hyper_edge__free(hyper_edge *edge)
{
    free(edge);
}

int hyper_graph__add_edge(hyper_edge *edge, hyper_graph *graph)
{
    if (graph->size == MAX_CLUSTER_SIZE)
        err("too many edges in hyper graph");
    edge->id = graph->size;
    graph->edge_list[graph->size++] = edge;
    return 1;
}

int find_hyper_edge_first_node(splice_graph *graph, identifier_t est_id, int *min_pos) {
    int i, pos, min_node=-1;
    *min_pos=1000;
    for (i=0; i<graph->size; i++) {
        pos = cprof_get_seq_first_location(graph->node_list[i]->profile, est_id, -1);
        if (pos != -1 && pos < *min_pos) {
            *min_pos = pos;
            min_node = i;
        }
    }
    return min_node;
}

*****
* The given est (est_id) is assumed to be in the source profile.
* This function answer 1 only if the current pass of the est in the
* profile (the one begins with pos) continues in the target profile.
*****
int hyper_edge_passes(int source_idx, int target_idx,
    identifier_t est_id, int pos, splice_graph *graph) {
    cprof prof1 = graph->node_list[source_idx]->profile;
    cprof prof2 = graph->node_list[target_idx]->profile;
    int est_pos1, est_pos2;

    est_pos1 = cprof_get_id_pos(prof1, prof1->len-1, est_id, pos);
    est_pos2 = cprof_get_id_pos(prof2, 0, est_id, pos);
    if (est_pos1 == -1) {
        /* the est ends in this node */
        return 0;
    }
    return (((est_pos2 - est_pos1) < 10) && (est_pos2 - est_pos1) > 0);
}

void create_sub_edges(hyper_graph *graph)
{

```

hyper_graph.c

```

hyper_edge *edge;
splice_node *node;
int i, nbr;

for(i=0; i<graph->s_graph->size; i++) {
    node = graph->s_graph->node_list[i];
    for (nbr=0; nbr < node->out_degree; nbr++) {
        edge = hyper_edge_new();
        edge->est = -1;
        edge->graph = graph->s_graph;
        edge->len = 2;
        edge->nodes = malloc(2*sizeof(int));
        edge->nodes[0] = i;
        edge->nodes[1] = node->out_neighbors[nbr].idx;
        hyper_graph__add_edge(edge, graph);
    }
}

void find_edge_relationship(hyper_graph *graph)
{
    int i, j, k, rel, N=graph->size;
    hyper_edge *edge1, *edge2;

    for(i=0; i<N; i++) {
        edge1 = hyper_graph__get_edge(i, graph);
        for(j=0; j<N; j++) {
            if (i == j) {
                continue;
            }
            edge2 = hyper_graph__get_edge(j, graph);
            for(k=0; k<edge1->len; k++) {
                if (((rel = path_compare(edge1, edge2, k)) == 1) && k>0) {
                    dual_edge__init(&(edge1->neighbors[edge1->out_degree++]),
                        edge1, edge2, edge1->len-k);
                    break;
                }
                else if (rel == -1) {
                    edge1->contained_edges[edge1->contained_num++] = edge2;
                    break;
                }
            }
        }
    }

    /* Compare edge1 to edge2 from START in edge1.
    * if we reach a point that edge1[START+i] != edge2[i] return 0,
    * if edge2 ends before reaching this point we return -1.
    * if edge1 ends before reaching the above we return 1.
    */
    int path_compare(hyper_edge *edge1, hyper_edge *edge2, int start)
    {
        int idx1, idx2;

```

hyper_graph.c

```

for(idx1=start, idx2=0; idx2<edge2->len && idx1<edge1->len; idx1++, idx2++) {
    if (edge1->nodes[idx1] != edge2->nodes[idx2])
        return 0;
}
if (idx2 == edge2->len)
    return -1;
return 1;
}

int remove_contained_edges(hyper_graph *graph)
{
    int edge_idx, del_edge_idx, nbr_idx;
    int removed_removed_num = 0;
    hyper_edge *del_edge, *edge;

    for (del_edge_idx=graph->size-1; del_edge_idx>=0; del_edge_idx--) {
        removed = 0;
        del_edge = hyper_graph__get_edge(del_edge_idx, graph);
        if (del_edge->est != -1) {
            for (edge_idx=0; edge_idx<graph->size; edge_idx++) {
                edge = hyper_graph__get_edge(edge_idx, graph);
                for (nbr_idx=0; nbr_idx<edge->contained_num; nbr_idx++) {
                    if (edge->contained_edges[nbr_idx] == del_edge) {
                        removed_num++;
                        removed = 1;
                        hyper_graph__remove_edge(del_edge, graph);
                        break;
                    }
                }
                if (removed) break;
            }
        }
        return removed_num;
    }

    void hyper_graph__remove_edge(hyper_edge *del_edge, hyper_graph *graph)
    {
        int i, nbr;
        hyper_edge *edge;

        for(i=0; i<graph->size; i++) {
            edge = hyper_graph__get_edge(i, graph);
            for (nbr=0; nbr<edge->out_degree; nbr++) {
                if (edge->neighbors[nbr].target == del_edge) {
                    dual_edge__copy(&(edge->neighbors[nbr]), &edge->neighbors[--(edge->out_deg
re)]);
                }
            }
            for (nbr=0; nbr<edge->contained_num; nbr++) {
                if (edge->contained_edges[nbr] == del_edge) {
                    edge->contained_edges[nbr] = edge->contained_edges[--(edge->contained_nu
m)];
                }
            }
            graph->edge_list[del_edge->id] = graph->edge_list[--(graph->size)];
            graph->edge_list[del_edge->id]->id = del_edge->id;
        }
    }
}

```

hyper_graph.c

```

hyper_edge___free(del_edge);
}

int hyper_graph___intersection_len(int edge1_idx, int edge2_idx, hyper_graph *graph)
{
    int nbr;
    hyper_edge *edge;
    edge = hyper_graph___get_edge(edge1_idx, graph);
    for(nbr=0; nbr<edge->out_degree; nbr++)
        if(edge->neighbors[nbr].target->id == edge2_idx)
            return (edge->neighbors[nbr]).relationship;
}

int hyper_graph___contains(int idx1, int idx2, hyper_graph *graph)
{
    int nbr;
    hyper_edge *edge;
    edge = hyper_graph___get_edge(idx1, graph);
    for(nbr=0; nbr<edge->contained_num; nbr++)
        if(edge->contained_edges[nbr]->id == idx2)
            return 1;
    return 0;
}

void hyper_graph___print(hyper_graph *graph)
{
    int i;
    printf("*****\n");
    printf("Hyper Graph: \n size: %d\n", graph->size);
    for(i=0; i<graph->size; i++) {
        printf("\n");
        hyper_graph___print(hyper_graph___get_edge(i, graph));
    }
    printf("*****\n");
}

/***** hyper_edge *****/
hyper_edge *hyper_graph___get_edge(int idx, hyper_graph *graph) {
    return graph->edge_list[idx];
}

splice_node *hyper_edge___last_node(hyper_edge *edge)
{
    return edge->graph->node_list[edge->nodes[edge->len-1]];
}

/*
int hyper_edge___type(hyper_edge *edge)
{
    return hyper_edge___last_node(edge)->type;
}
*/

```

```

int hyper_edge___is_initial(hyper_edge *edge)
{
    return edge->graph->node_list[edge->nodes[0]]->type & INITIAL;
}

int hyper_edge___is_end(hyper_edge *edge)
{
    return hyper_edge___last_node(edge)->type & ENDING;
}

int hyper_edge___last_node_marked(hyper_edge *edge)
{
    return hyper_edge___last_node(edge)->marked;
}

int hyper_edge___mark_last_node(hyper_edge *edge)
{
    return hyper_edge___last_node(edge)->marked = 1;
}

void hyper_edge___print(hyper_edge *edge)
{
    int i;
    printf("Edge %d: \n", edge->id);
    printf("len: %d, est: %d, out degree: %d\nPath: ", edge->len, edge->est, edge->out_degree);
    for(i=0; i<edge->len; i++) {
        printf("%d ", edge->nodes[i]);
    }
    printf("\nNeighbors: ");
    for(i=0; i<edge->out_degree; i++) {
        printf("%d %d", edge->neighbors[i].target->id, edge->neighbors[i].relationship);
    }
    printf("\n");
}

int hyper_edge___check(hyper_edge *edge)
{
    int i;
    for(i=0; i<edge->len-1; i++) {
        if(!splice_graph___is_directed_edge(edge->graph, edge->nodes[i], edge->nodes[i+1])) {
            printf("edge %d passes from %d to %d (not connected)",
                edge->id, edge->nodes[i], edge->nodes[i+1]);
            return 1;
        }
    }
    return 0;
}

/**** dual_edge ****/
void dual_edge___init(dual_edge *de, hyper_edge *source, hyper_edge *target, int re)
{
    de->source = source;
    de->target = target;
    de->relationship = rel;
}

void dual_edge___copy(dual_edge *target, dual_edge *source)

```

```
(
    target->source = source->source;
    target->target = source->target;
    target->relationship = source->relationship;
)

void dual_edge___free(dual_edge *de)
{
    free(de);
}
```



```
/*
 * Additional static identifiers for assembly program
 *
 * Can be used to identify features compiled into the assembly program.
 */
/*
 * $Log: id.c,v $
 * Revision 1.4 1998/06/18 14:42:47 ariels
 * Strip "#ifdef CPROF_ALIGN" and "#ifdef IMPROVE_CPROF" lines.
 *
 * Revision 1.3 1998/02/11 16:26:03 ariels
 * * better formatting for the features[] string.
 * * identify IMPROVE_CPROF
 *
 * Revision 1.2 1998/01/11 07:37:40 ariels
 * Add float/fixed-point scores identifier
 *
 * Revision 1.1 1997/12/17 11:59:08 ariels
 * Initial revision
 */
```

```
static char rcsid[] = "$Id: id.c,v 1.4 1998/06/18 14:42:47 ariels Exp $";
```

```
static char features[] = "$Keyword: Assembly ("
```

```
"#DEC"
```

```
"#SUN"
```

```
"#if defined(__sun)"
```

```
"#elif defined(__sgi)"
```

```
"#SGI"
```

```
"#elif defined(__linux)"
```

```
"#Linux"
```

```
"#endif"
```

```
" OS)"
```

```
"
```

```
" cprof2cpref"
```

```
"#ifdef DEBUG_LINEAR"
```

```
" [debug_linear]"
```

```
"#endif"
```

```
"#ifdef USE_INTEGER_COSTS"
```

```
" integer_costs"
```

```
"#endif"
```

```
"#ifdef USE_FLOAT_SCORE"
```

```
" float_scores"
```

```
"#else"
```

```
" fixed-point_scores"
```

```
"#endif"
```

```
" local_profile_improvements"
```

```
"
```

```
"#ifdef PENALTY"
```

```
" penalty"
```

```
"#endif"
```

```
"#ifdef COMPACTIZATION"
```

```
" compactization"
```

```
"#endif"
```

```
"#ifndef NDEBUG"
```

```
" asserts"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```

```
"
```



```

/* implementation of i/o functions. */
/*
$Log: io.c,v $
Revision 1.60 1998/06/30 22:41:50 avner
check_repeats=v now means a printout of the repeats will be produced for
the viewer.
Revision 1.59 1998/06/18 14:42:47 ariels
Strip "#ifdef CPROF_ALIGN" and "#ifdef IMPROVE_CPROF" lines.
Revision 1.58 1998/06/17 07:37:52 eyal
Fix memory leak in read_sequences, when the contig is too big.
Revision 1.57 1998/06/15 08:15:16 eyal
Add function print_ests_hyper_edge_alignment and a call for it
Revision 1.56 1998/05/15 10:51:04 avner
adding the directionality of the sequence when annotated
('write_cluster_output').
Revision 1.55 1998/05/13 14:54:21 eyal
change read_sequences to read all read_sequences when assembly_unit=a
Revision 1.54 1998/05/13 13:22:58 eyal
1. Add a call to graph_print_cprefs()
2. Add a call to find_ests_paths(graph) and est_table_verify_est_path
Revision 1.53 1998/05/04 14:12:42 eyal
free components in cluster_post_process
Revision 1.52 1998/05/04 12:37:14 eyal
Add functions print_connective_components and mark_ests_components.
2. add a call for these function and forsplice_graph_find_connective_compon
ents, when a disconnected graph is encountered
Revision 1.51 1998/05/03 11:44:15 ariels
Add "ENDCONTIG" on last line of graph file record for singletons.
Revision 1.50 1998/04/25 17:05:49 avner
1. make need_rpmode be int and not char.
2. deal with cons map in print_alignments functions, in such a way that the
<len> location is not used.
Revision 1.49 1998/04/25 16:28:47 avner
mark polyT in print_alignment.
Revision 1.48 1998/04/24 21:50:37 avner
solve few technicalities when dealing with singletons (calloc instead
of memset is the more conspicuous).
Revision 1.47 1998/04/24 12:48:20 avner
'rich_fast_read_seq' is called with the ASSEMBLY_MODE parameter.
Revision 1.46 1998/04/23 21:12:12 avner
make N align with '.' only when it is not against a gap.
Revision 1.45 1998/04/23 20:47:21 avner
make N in bottom sequence align with : to anything (gap also), so that it will

```

```

* appear orange.
* Revision 1.44 1998/04/23 14:01:06 eyal
* Add field #TY TRS to the transcript header
* Revision 1.43 1998/04/22 12:39:56 ariels
* Incorporate 1.31.1 branch for colouring inverted repeats.
* Revision 1.42 1998/04/20 13:54:26 ariels
* Move clone-length on transcript calculation to print_support(), where
* it can use the initial verification that a clone pair indeed lies
* along a transcript.
* Revision 1.41 1998/04/20 07:33:39 eyal
* Fix bug in get_transcript_alignment
* Revision 1.40 1998/04/18 22:16:03 avner
* changes due to the different way in which we deal with the 'dirty'
* fields.
* Revision 1.39 1998/04/15 14:45:28 ariels
* Calculate length of each clone pair along each transcript. New
* routine get_transc_loc() is used for this; given a location on the
* consensus, a map of a transcript, and the consensus map, it returns
* the aligned transcript location, or -1 if that location isn't covered
* by the transcript.
* Revision 1.38 1998/04/15 10:14:20 eyal
* 1. add function tissue_hash_fill
* 2. add #ES and #rn to transcripts headers
* 3. use write_transcripts_headers, write_transcripts and tissue_hash_fill,
* when in rp_mode with a cycle
* 4. change pass from short to int in test_node
* Revision 1.37 1998/04/13 08:21:42 eyal
* 1. replace tissue_hash_print with tissue_hash_print_on_string
* 2. change the use of char *transc_db to FILE *transc_file
* 3. write functions write_transcripts and write_transcripts_headers
* and use them
* Revision 1.36 1998/04/12 12:45:36 eyal
* 1. finish avner's work on the cview file
* 2. change type of map from char* to int* (all over the file)
* Revision 1.35 1998/04/12 08:35:42 avner
* This version does not compile! Work in progress for adding transcripts to
* cview file. Other than that:
* 1. make tissue printing for transcripts with multiplicity (a change
* in the hash-table functions).
* 2. print tissues without underscores.
* 3. dont print profile for singletons.
* Revision 1.34 1998/04/09 17:07:51 eyal
* add a parameter to 'read_sequences' called 'assembly_unit'. When it's 'u', th
e unit we read here
* is a cluster and not a contig as in the other cases.
* Revision 1.33 1998/04/09 12:34:19 eyal
* 1. Add polyA marking in the pfs format.
* 2 replace compilation flag ONLY_TRANS with the regular rp_mode.

```

- * Revision 1.32 1998/03/29 21:05:52 avner
- * Improve 'print_keywords'. It does not print anything if no keywords-file
- * supplied, and some peculiarity were being fixed.
- * Improve 'predict_clone_length', with a care to the new flag 'dirty_tails'.
- * Revision 1.31.1.1 1998/04/14 15:18:07 ariels
- * Paint inverted repeats beautifully on output.
- * Revision 1.31 1998/03/24 11:02:49 eval
- * 1. Change the order of build_transcripts and build_consensus
- * 2. Add a preprocessor flag ONLY_TRANS which must exist if you want
- * to run in rp_mode and post_process - it does not attempt to build
- * consensus but just writes the transcripts.
- * Revision 1.30 1998/03/17 16:26:33 avner
- * outputting tails (suspected in being dirty) in the final alignments of
- * sequences vs. the consensus with a special color.
- * add clone length estimation (by the distance in the consensus - not perfect
- * of course) and previous information we have from the input's annotation.
- * Revision 1.29 1998/03/08 21:29:30 avner
- * fix bug of strand indication.
- * add clone-length prediction mechanism.
- * Revision 1.28 1998/03/08 11:50:10 ariels
- * Can't get ins/del/mis tags from hilltops data when processing
- * singletons (there isn't any hilltops data, and they're always 0%
- * anyway!).
- * Revision 1.27 1998/02/23 15:46:28 ariels
- * Add ins/del/mismatch percentages to the (EMBL) output file.
- * Revision 1.26 1998/02/23 11:11:52 ariels
- * Fix spelling of 'consensus'.
- * Revision 1.25 1998/02/22 21:47:04 avner
- * distinguish between rna and est in the error (of alignments with consensus)
- * report, in order to learn some interesting statistics.
- * Revision 1.24 1998/02/22 17:16:31 ariels
- * Use new error and warning reporting facility.
- * Revision 1.23 1998/02/21 22:48:19 avner
- * changes due to different repeats-observing mechanism, and mostly adjusting
- * to the changing needs of 'print_contig_online_report'.
- * Revision 1.22 1998/02/19 16:08:20 eval
- * Adding a call for rp_mode_build_transcripts
- * Revision 1.21 1998/02/19 11:52:32 ariels
- * Write graph structure file.
- * Revision 1.20 1998/02/17 16:11:43 ariels
- * Report warnings in the EMBL file.
- * Report cluster name in the EMBL file.
- * Track down and fix warning message (chimeric transcripts) which wasn't going
- * through the proper channels.
- * Minor source layout fixes.

- * Revision 1.19 1998/02/11 15:29:18 avner
- * add function 'compare_consensus'.
- * Revision 1.18 1998/02/09 13:15:09 ariels
- * Fixed cluster/contig defaults bug in read_sequences().
- * Revision 1.17 1998/02/08 13:33:45 ariels
- * Separated rich-Fasta I/O routines into rf.[ch]
- * Revision 1.16 1998/01/21 07:53:04 ariels
- * Put err() in scope of prototype (add #include "error.h")
- * Revision 1.15 1998/01/14 10:06:20 eval
- * Add support for rna_test_mode.
- * Revision 1.14 1998/01/12 16:03:07 ariels
- * Hack print_alignment_genweb() to print gaps as DOTS ('.') instead of
- * DASHES ('-'), to work around a builtin limitation of genweb.
- * Revision 1.13 1997/12/29 10:27:41 ariels
- * Fixed idiotic bug in deal_with_singletons() which caused the "#LN"
- * field of singleton pseudo-transcripts to be printed as "N" instead.
- * Revision 1.12 1997/12/22 07:49:59 ariels
- * Change read_sequences to return to contig_id (cid) in the form
- * CLUSTER.CONTIG when reading GenBank >= 104 format rich-Fasta input
- * files.
- * Revision 1.11 1997/12/20 23:06:19 avner
- * 'found_repeat_in_consensus' turns to be global.
- * Revision 1.10 1997/12/20 21:26:20 avner
- * function 'deal_with_singletons' (from main) to here.
- * Revision 1.9 1997/12/18 09:58:19 eval
- * Adding a color print of repeats on consensus (when check_repeats=m is used)
- * Rudimentary support for cluster_view output (no features, est warnings
- * don't make it to the docs).
- * Revision 1.8 1997/12/17 15:35:15 ariels
- * Rudimentary support for cluster_view output (no features, est warnings
- * don't make it to the docs).
- * Revision 1.7 1997/12/17 13:02:02 ariels
- * From GenBank 104 onwards, rich-Fasta should have a "#CU" cluster
- * (formerly metaccluster) field and a "#CN" contig (formerly cluster)
- * field, instead of just the old "#CU" field. Parse this field when
- * reading rich-Fasta.
- * Revision 1.6 1997/12/14 14:21:04 ariels
- * Added 'splicemarks' flag, which adds \$P marks to the rich-Fasta
- * transcripts (in transc.db), which give the exact position of each
- * proposed splice location. Specifically, at the end of the extent of
- * each node in the transcript we place a '#SP pos <out:in>' marker,
- * where pos is the position in the transcript, out the out-degree of the
- * node we're leaving, and in the in-degree of the node we're entering.
- * Default (in prologue()) is splicemarks=n, which leaves the transcript
- * database in the old format.
- * Revision 1.5 1997/12/10 22:09:01 avner

```

* *** empty log message ***
*
* Revision 1.4 1997/11/30 13:09:04 ariels
* Straightened tissue table in EMBL file
*
* Revision 1.3 1997/11/30 09:55:58 ariels
* Cleaned logic when printing chimeras
*
* Revision 1.2 1997/11/30 07:59:42 ariels
* Added ChangeLog comment and static rcsid string.
*
static char rcsid[]="$Id: io.c,v 1.60 1998/06/30 22:41:50 avner Exp $";

#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>

#include "rf.h"
#include "io.h"
#include "general.h"
#include "est_table.h"
#include "error.h"

#include "splice_graph.h" /* for est support stuff */
#include "repeats.h"
#include "hyper_graph.h"

extern int cutoff_bound;
extern int found_repeat_in_consensus;
extern double straight_repeat_max_sim;
extern char p_est_error_out_file[keyword_file[]];
extern char **transcripts;
extern int gb_ver, need_rpmode;
extern char *prefix, *organism, date[];
extern char rp_mode, dirty_tails;
static void print_trivial_alignments(FILE *out, char **int **, int n,
char *, int *, int *, char *, char *);

static char *replace_char1_by_char2(char *, char, char);

static void assembly_type_report(FILE *, splice_graph *, int, int, int);

static void print_repeats_on_consensus(FILE *out, char **transcripts,
int **transc_map, int n,
char *consensus, int *map, int *node_order,
char *cl_no, char *colors, int);

static void compare_consensus(char *seq);
static void get_transcript_alignment(FILE *fp, char *consensus, int *map,
int *transc_map, int *node_order);
static void write_transcripts_headers(char **headers, char *contig_name,
char *cluster_name, int **tr_map,
char **transcripts, int num_transcripts,
tissue_hash *tissue, splice_graph *graph);

```

io.c

```

static void write_transcripts(FILE *fp, char **tr_headers, char **transcripts,
int num_transcripts);
static void tissue_hash_fill(tissue_hash *tis, int **transc_map, int transc_num,
er);
static void predict_clone_length(int graph_size, char *cu_name, char *cn_name,
int, int **, int *);
void mark_est_components(splice_graph *graph, int *components);
void print_connective_components(splice_graph *graph, int *components,
int comp_num);
void print_est_hyperm_alignment();

extern char check_repeats, cmp_res[], p_error_data, post_process;

static char *est_noun(int i)
{
    static char est_name[]="est";
    static char rna_name[]="rna";
    static char seq_name[]="seq";
    if (est_table__is_rna(i))
        return rna_name;
    else if (est_table__is_est(i))
        return est_name;
    else
        return seq_name;
}

void read_sequences(fasta_file ff, char *contig_name, char *cluster_name, char asse
mbly_unit)
{
    static rich_fasta_seq_ptr rfp = NULL;
    static int first = 1;
    int real_size=0;
    int too_big=0;

    contig_name[0]='\\0';

    if(first) {
        rfp = rich_fasta_read_seq(ff, ASSEMBLY_MODE);
        first = 0;
    }
    if(!rfp) return;
    if (rfp->cluster)
        strcpy(contig_name, rfp->cluster);
    if (rfp->meta_cluster)
        strcpy(cluster_name, rfp->meta_cluster);
    else
        cluster_name[0] = '\\0';

    while (rfp && ((strcmp(contig_name, rfp->cluster)==0) ||
        (strcmp(rfp->cluster, "UNKNOWN") == 0 &&
        strcmp(contig_name, rfp->meta_cluster) == 0) ||
        (assembly_unit=='u' && strcmp(cluster_name, rfp->meta_cluster) =
=0) ||
        assembly_unit=='a'))
    {
        real_size++;
        if (est_table__size() < MAX_CLUSTER_SIZE)
            est_table__add (rfp);
        else {

```

io.c

```

rich_fasta_seq_free(rfp);
too_big=1;
}
rfp = rich_fasta_read_seq(ff,ASSEMBLY_MODE);
}
if(too_big)
    printf("Warning: contig %s is too big (%d)\n",
           cluster_name,contig_name,real_size);
/* reset_graph_edges (est_table__size()); */
}

static void print_keywords (char *cl_no)
{
    int i,j;
    FILE *fp;
    int yf = 0;
    if (strcmp(keyword_file,"none") != 0 && (fp = fopen(keyword_file,"a")))
    {
        fprintf(fp,"%s ",cl_no);
        for(i=0;i<5;i++) {
            for (j=0;est_table__keywords(i,j);j++) {
                switch (i) {
                    case 0:
                        fprintf(fp,"#ID ");
                        break;
                    case 1:
                        fprintf(fp,"#CL ");
                        break;
                    case 2:
                        fprintf(fp,"#TI ");
                        break;
                    case 3:
                        fprintf(fp,"#LB ");
                        break;
                    case 4:
                        fprintf(fp,"#CH ");
                        break;
                }
                fprintf(fp,"%s ",est_table__keywords(i,j));
            }
            fprintf(fp,"\n");
            fclose(fp);
        }
    }

/* Create a list of all alignments between consensus and ests
*/
hilltop **get_est_alignments(char *consensus)
{
    hilltop **res;
    int i;

    if ((res = malloc(est_table__size() * sizeof(hilltop *))) == NULL)
        err("memory failure allocating %d hilltops", est_table__size());
}
    
```

io.c

```

for(i=0; i<est_table__size(); i++) {
    res[i] = get_six8_alignment(consensus, est_table__seq(i), 0);
    est_table__mark_alignment_range(i,res[i]->x0,res[i]->xt);
}
return res;
}

void free_est_alignments(hilltop **hts)
{
    int i;
    if (hts) {
        for(i=0; i<est_table__size(); i++)
            destroy_hilltop(&(hts[i]));
        free(hts);
    }
}

/*
 * given the alignment data, the consensus and the est, report which
 * nodes of the graph the est goes through.
 *
 * ariels 11/11/97
 */

static void test_nodes(hilltop ht, int *map,
                      int est_no, splice_graph *graph, int *cons_map)
{
    int i;
    int idx;
    int f;
    int match, len;
    int node;
    int pass[MAX_NODES+1];
    extern int indep_node_len;

    idx = 0;
    f = ht.x0;

    node = -1;
    for(i=0; i<ht.len; i++) {
        if (node != cons_map[map[f]]) {
            if (node != -1) {
                /* Decide if est passes through node */
                if (len >= indep_node_len)
                    if (3*match > 2*len)
                        pass[idx++] = node; /* passes through node */
                else if (3*match > len)
                    record_warn("Possible error for est %d "
                                "passing through node %d (%d/%d)\n",
                                est_no, node, match, len);
            }
            match = len = 0;
            node = cons_map[map[f]];
        }
        if (ht.x[i] != '-')
            f++;
        len++;
    }
}
    
```

io.c

A-104

```

    if (ht.diff[i] == '|')
        match++;
    }

    /* do last node */
    if (len >= indep_node_len)
        if (3*match > 2*len)
            pass[idx++] = node;
        else if (3*match > len)
            record_warn("Possible error for est %d",
                "passing through node %d (%d/%d)\n",
                est_no, node, match, len);
    pass[idx++] = -1; /* terminate list */

    /* Post-process node transitions of this est */
    for (i=0; i<idx-2; i++)
        if (!splice_graph__is_directed_edge(graph, pass[i], pass[i+1]))
            record_warn("Est %d passes from node %d to non-neighbouring node %d\n",
                est_no, pass[i], pass[i+1]);
    else {
        /* add est to list of ests going along this edge */
        splice_graph__add_est_between_nodes(graph, est_no, pass[i], pass[i+1]);
    }
    est_table__set_node_path(est_no, pass, idx);
}

/*
 * return array with "number" (0/1/2="many") of paths between nodes in
 * splice graph.
 */
static int *calc_paths(splice_graph *graph)
{
    int *mat, *mat2, *tmp;
    int i, j, k, l;
    int sum;

    if ((mat = splice_graph__get_matrix(graph)) == NULL ||
        (mat2 = calloc(graph->size*graph->size, sizeof(int))) == NULL) {
        if (mat) free(mat);
        err("memory failure in calc_paths for %dx%d adjacency matrices",
            graph->size, graph->size);
    }

    for (l = 2*graph->size; l > 0; l >= 1) {
        /* floor(lg2(graph->size))+1 times */
        /* "square" matrix, counting only 0/1/2="many" */
        for (i=0; i<graph->size; i++)
            for (k=0; k<graph->size; k++) {
                for (j=0, sum = 0; j<graph->size; j++)
                    sum += mat[i*graph->size+j] * mat[j*graph->size+k];
                mat2[i*graph->size+k] = (sum > 1) ? 2 : sum;
            }
        tmp = mat2;
        mat2 = mat;
        mat = tmp;
    }
}

```

```

    }

    free(mat2);
    return mat;
}

/*
 * check if a "bridge" from start to end is enough to support a
 * transcript.
 *
 * In practice, a bridge should either be from the first to last nodes of
 * an est, or from the first node of the 5' est of a clone to the last
 * node of the 3' est of that clone, when there is a unique path
 * connected the 2 ests.
 */
static int bridge_supports_transcript(int start, int end, int *t_map,
    splice_graph *graph)
{
    int j;

    /* Get initial segment */
    for (j=0; t_map[j] != -1; j++)
        if (graph->node_list[t_map[j]]->in_degree > 1)
            break;

    if (j-1 < start)
        return 0;

    /* Get final segment */
    for (j=end; t_map[j] != -1; j++)
        if (graph->node_list[t_map[j]]->out_degree > 1)
            break;

    return (t_map[j] == -1);
}

/*
 * Translate a location in the consensus to a location in the transcript;
 * if that location doesn't match anything in the transcript, return -1.
 * Probably won't work for cyclic graphs unless the "consensus" actually
 * manages to cover the transcript.
 */
static int get_transc_loc(int cons_loc, int *transc_map, int *cons_map)
{
    int i;
    int transc_loc;
    int transc_node;
    int in_node;

    in_node = 0;
    transc_node = -1;
    transc_loc = -1;

    for (i=0; i<cons_loc; i++) {
        if (in_node) {
            if (cons_map[i] == transc_map[transc_node])
                ++transc_loc;
        }
    }
}

```



```

else
    in_node = 0;
}
if ( (! in_node && cons_map[i] == transc_map[transc_node+1]) ) {
    in_node = 1;
    ++transc_node;
    ++transc_loc;
}
}
return in_node ? transc_loc - 1;
}

```

```

/* Search for est in transcript map; store start and end nodes, and
 * return indication of success
 */

```

```

static int get_est_bounds_in_transcript(int est_no, int *t_map,
int *start, int *end)

```

```

{
    int j, k;

    /* find start of in_node list in trans_map[transc_no] */
    for(j=0; t_map[j] != -1; j++)
        if (t_map[j] == est_table__node_path_pos(est_no, 0))
            break;
    if (t_map[j] == -1)
        return 0;
    /* not found */
    *start = j;

    for(k=0; est_table__node_path_pos(est_no, k) != -1; j++, k++)
        if (t_map[j] != est_table__node_path_pos(est_no, k))
            break;
    if (est_table__node_path_pos(est_no, k) != -1)
        return 0;
    /* not found */

    *end = j-1;
    return 1;
}

```

```

/* print_support -- print est subsequence and support data
 *
 * Check each transcript to see if the given est agrees with it. Print
 * information about this agreement, then for each agreeing transcript
 * check to see if the est supports it. It is an internal error for an
 * est to support more than one transcript!
 *
 * ariels 11/11/97
 */
int print_support(FILE *fp, char *cl_no,
int *cons_map, int **transc_map, int n, tissue_hash *tis,
splice_graph *graph, int *unique_path)
{
    int est_no;
    int clone_est;

    /* this est's clone-mate, if it is a 5' end */
    /* and has a 3' end clone-mate; otherwise -1 */

```

io.c

```

/* # of supported transcripts (<=1) */
/* transcript we're now looking at */
/* first & last nodes in transcript that */
/* this est passes through */
/* ditto for this est's clone-mate */
/* # of transcripts agreeing both with */
/* est and with its clone-mate */

enum {
    e_disagree = 0, e_agree = 1, e_support = 2, e_eol = 3
} *est_transc_tbl;
enum { e_rna, e_est, e_done } pass;
short have_ximeras = 0; /* 1 if we should print table of chimeras */

/* Code for agreement between est and transcript */
short *est_supports; /* 1 if est supports some transcript */
int num_supported;

```

```

if ((est_supports = calloc(est_table__size(), sizeof(short))) == NULL)
    err("memory failure in print_support for %d",
        est_table__size());
if ((est_transc_tbl =
    malloc(est_table__size() * n * sizeof(*est_transc_tbl))) == NULL)
    err("memory failure in print_support for %dx%d", est_table__size(), n);
num_supported = 0;

for(est_no = 0; est_no < est_table__size(); est_no++) {
    supported = 0;
    /* search for single-est support */
    for(transc_no = 0; transc_no < n; transc_no++) {
        est_transc_tbl[est_no*n + transc_no] = e_disagree;
        if (! get_est_bounds_in_transcript(est_no, transc_map[transc_no],
            &first, &last))
            continue;

```

```

/* est agrees with transcript */
est_transc_tbl[est_no*n + transc_no] = e_agree;

/* Store tissue from matching est (doesn't really belong
here, but this is the only place we find which transcripts
the est agrees with! */
if (tis && est_table__seq_data(est_no) -> tissue)
    (void)tissue_hash_intern(tis[transc_no],
        est_table__seq_data(est_no) -> tissue);

if (bridge_supports_transcript(first, last,
    transc_map[transc_no], graph)) {
    /* fprintf(fp, "CC %s %10s SUPPORTS transcript %s%d\n",
        est_noun(est_no), id(est_no), prefix, gb_ver, cl_no, transc_no);
    */
    est_transc_tbl[est_no*n + transc_no] = e_support;
    ++supported;
}
}
if (supported > 1)
    err("Too many supported transcripts for est %d", est_no);
num_supported += supported;
est_supports[est_no] = supported;

```

io.c


```

if (est_table__strand(est_no) != est_table__strand(clone_est)) {
    int five_p = est_table__strand(est_no)==1 ? est_no : clone_est;
    int three_p = est_table__strand(est_no)==1 ? clone_est : est_no;
    int start = get_transc_loc(est_table__get_start_in_cons(five_p),
        transc_map[transc_no], cons_map);
    int end = get_transc_loc(est_table__get_end_in_cons(three_p),
        transc_map[transc_no], cons_map);
    if (start != -1 && end != -1) {
        start -= est_table__trim3(five_p);
        end += est_table__trim3(three_p);
        printf(">>RR &LW & #GB & #CL &LW\n", transc_no,
            end-start, est_table__clone_length(five_p, three_p),
            est_table__clone_name(five_p));
    }
}

if (est_supports[est_no] || est_supports[clone_est])
    continue;

/* Try to get support from clone */
if ((clone_first <= last) ||
    unique_path[last*graph->size + clone_first] == 1) &&
    bridge_supports_transcript(first, clone_last,
        transc_map[transc_no], graph)) {
    fprintf(fp, "CC %s %10s and %s %10s SUPPORT transcript %s%d.%s.%d\n",
        est_noun(est_no), id(est_no),
        est_noun(clone_est), id(clone_est),
        prefix, gb_ver, cl_no, transc_no);
    ++supported;
}

if (! clone_ok)
    record_warn("clone pair ests %d and %d match no transcript!\n",
        est_no, clone_est);
if (supported > 1)
    err("Too many supported transcripts for est pair %d.%d",
        est_no, clone_est);
num_supported += supported;
}

free(est_supports);
free(est_transc_tbl);
return num_supported;
}

/* cluster_view output
*
* This should probably be in a separate file.
*
* ToDo:
*
* + Improve documentation (warnings for each est should appear in its
*   'DOC' fields, before the tissue list).
* + Mark or propagate features ('FEAT' fields).
*/

```

io.c

```

static char cv_fmt_ver[] = "assembly 1.0";
#define GB104 if (gb_ver < 104) return
void cv_write_header(FILE *fp)
{
    GB104;
    fprintf(fp, "VER %s\n", cv_fmt_ver);
}

static hilltop *cv_separate_hilltop(hilltop ht)
{
    hilltop *res;

    create_hilltop(&res);
    copy_hilltop(ht, res);
    improve_six18_result(res, SEPARATE_HILLTOPS);
    return res;
}

void cv_write_contig(FILE *fp, char *cl_no, char *consensus,
    int num_transcripts, int *cons_map,
    char **tr_headers, int **tr_map, int *node_order,
    hilltop **hts)
{
    int i;
    char *tis;
    hilltop *ht, *p;

    GB104;
    fprintf(fp, "ID %s\n", cl_no);
    fprintf(fp, "CLUSTER %s\n", est_table__seq_data(0)->meta_cluster);
    fprintf(fp, "LENGTH %d\n", strlen(consensus));

    /* *** INSERT ADDITIONAL DOCUMENTATION (WARNINGS) HERE *** */
    fprintf(fp, "DOC");
    for(i=0; tis=est_table__keywords(TISSUE,i); i++)
        fprintf(fp, "%s %s", i ? " " : "", tis);
    putc('\n', fp);

    for(i=0; i<num_transcripts; i++) {
        fprintf(fp, "HEADER %s", tr_headers[i]);
        get_transcript_alignment(fp, consensus, cons_map, tr_map[i], node_order);
    }
    for(i=0; i<est_table__size(); i++) {
        fprintf(fp, "HEADER %s", est_table__seq_data(i)->header);

        if (hts) {
            ht = cv_separate_hilltop(*hts[i]);
            for(p=ht; p; p=p->next)
                fprintf(fp, "ALIGN %d.%d.%d\n", p->x0, p->xt, p->y0, p->yt);
        }

        /* *** INSERT FEAT (FEATURE) KEYWORD LINES HERE *** */
        destroy_hilltop_list(&ht);
    }
    else
        /* Dummy case for stand-alone clusters */
        fprintf(fp, "ALIGN %d.%d.%d\n",
            0, strlen(consensus)-1, 0, strlen(consensus)-1);
}

```

io.c

```

)
fprintf(fp, "\\n");
}

/* Graph/profile output
* Output to a file containing (nearly) the entire graph structure,
* transcripts, etc.
*/
void graph_write_record(FILE *fp, splice_graph *graph,
                        char *cluster, char *contig,
                        int n_transc, int **transc_map)
{
    int i, j;

    fprintf(fp, "CONTIG %s.%s\n", cluster, contig);

    /* Place DOC fields for contig here! */

    /* Print graph structure */
    fprintf(fp, "NODES %d\n", graph->size);

    /* Print edges. These will be hyperedges in future */
    for(i=0; i<graph->size; i++)
        for(j=0; j<graph->node_list[i]->out_degree; j++)
            fprintf(fp, "EDGE %d %d\n", i, graph->node_list[i]->out_neighbors[j].idx);

    /* Print node contents */
    for(i=0; i<graph->size; i++) {
        fprintf(fp, "NODE %d\n", i);
        fprintf(fp, "PROFILE !length %d\n", graph->node_list[i]->profile->len);
        cprof_num_print(fp, "PROFILE ", graph->node_list[i]->profile, 0, 0);
    }

    /* Dump transcripts */
    for(i=0; i<n_transc; i++) {
        fprintf(fp, "TRANSC");
        for(j=0; j<transc_map[i][j] != -1; j++)
            fprintf(fp, " %d", transc_map[i][j]);
        puts("\\n", fp);
    }

    /* FINISHED! */
    fprintf(fp, "ENDCONTIG\n");
    fflush(fp);
}

/* EMBL output
* Unlike before, this has been stripped of all the non-essential
* details. In particular, the consensus, transcripts, est alignments

```

```

* and all other "strategic" decisions about the output should already be
* computed before this is called.
*/
void write_cluster_output(char **transcripts, int n, char *consensus,
                          int consensus_len, int *node_order,
                          int unique_order,
                          char *contig_name, int map[],
                          int **transc_map,
                          tissue_hash *tissues,
                          splice_graph *graph,
                          hilltop **hts)
{
    int i, j, rev, a, c, g, t, o, counter;
    int insertions, deletions, mismatches, len, not_aligned;
    FILE *out;
    int tot_ins, tot_del, tot_mis, tot_main, tot_not, tot_len;
    double est_ins, est_del, est_mis, est_not;
    char *tis;
    int *unique_path_matrix;
    extern char cluster_name[];
    enum { e_rna, e_est, e_done } pass;

    found_repeat_in_consensus=0;

    est_table__create_keywords();
    if ((out = fopen(out_file, "a")) == NULL)
        error (1, -1, "Can't open output file %s", out_file);

    /* Cluster header few lines */
    fprintf (out, "ID %s cluster; RNA; EST; %d BP.\n",
             contig_name, consensus_len);
    fprintf (out, "AC %s\n", prefix, gb_ver, contig_name);
    fprintf (out, "DT %s (Rel. %d, created)\n", date, gb_ver);
    fprintf (out, "DE %s\n");
    fprintf (out, "OS %s\n", organism);

    /* PFS output */
    fprintf (out, "CC Statistics ! %d %d %d\n",
             n, est_table__est_num(), est_table__rna_num());

    /*
    * DO NOT CORRECT THE GRAMMAR IN THIS MESSAGE.
    * GENWEB ***DEPENDS*** ON THE APOSTROPHE BEING THERE!
    * FAILURE TO COMPLY WILL LEAD TO ***SILENT FAILURE*** IN GENWEB!!!
    */
    fprintf (out, "CC Alignment of %s&d %s to: it's cluster\n",
             prefix.gb_ver, contig_name);
    fprintf (out, "CC * * * * * \nCC \nCC * * * * *");

    fprintf (out, "CC Contig %s is in cluster %s\n", contig_name, cluster_name);

    /* Summarise what happened in this cluster */
    if (est_table__size() > 1) {
        /* EF - moving found_repeat_in_consensus up */
        if (check_repeats != 'n') {
            straight_repeat_max_sim = inverted_repeat_max_sim = 0;
            found_repeat_in_consensus = check_repeat(consensus, "consensus");
        }
    }

```

```

)
assembly_type_report(out, graph, get_cluster_volume(),
                    consensus_len, unique_order);
)

/* *** NOTE: You cannot have any warnings from this point on! They
will not be reported in the EMBL file if you do!! */
report_warn(out, "CC Warning: ");

/* List ets */
fprintf(out, "CC EST: ");
for (i = 0; counter = 0; i < est_table_size(); i++) {
    if (counter > 5) {
        fprintf(out, "\nCC EST: ");
        counter = 0;
    }
    if (est_table_is_est(i)) {
        fprintf(out, "%10.10s ", id(i));
        counter++;
    }
    fprintf(out, "\n");
}

/* List rnas */
fprintf(out, "CC RNA: ");
for (i = 0; counter = 0; i < est_table_size(); i++) {
    if (counter > 5) {
        fprintf(out, "\nCC RNA: ");
        counter = 0;
    }
    if (est_table_is_rna(i)) {
        counter++;
        fprintf(out, "%s ", id(i));
    }
    fprintf(out, "\n");
}

fprintf(out, "CC\n");

if (est_table_size() > 1)
{
    /* Print est support data */
    unique_path_matrix = calc_paths(graph);

    /* Print support */
    i = print_support(out, contig_name,
                    map, transc_map, n, tissues,
                    graph, unique_path_matrix);

    if (i)
        fprintf(out, "CC\n");
}

/* Collected list of tissues from all ets in the cluster */
fprintf(out, "CC TIS: ");
for (i = 0; counter = 0; tis = est_table_keywords(TISSUE, i); i++) {
    if (counter > 50) {
        fprintf(out, "\nCC TIS: ");
    }
}

```

io.c

```

counter = 0;
}
counter += strlen(tis);
fprintf(out, "%s", tis);
}

fprintf(out, "\nCC Sequence Strd Length/clean ins del mis
% Tissue ! Documentation\n");
/* EF -> */
/* This change is temporary - it should be replaced by a change in the GenWeb
*/

if (check_repeats == 'v' && (found_repeat_in_consensus & STRAIGHT_REPEAT))
    fprintf(out,
        "\nCC Repeats + %d\n", (int)strlen(consensus),
        "%s %s %s",
        "-", "-", "-");
if (check_repeats == 'v' && (found_repeat_in_consensus & INVERTED_REPEAT))
    fprintf(out,
        "\nCC Inverts + %d\n", (int)strlen(consensus),
        "%s %s %s",
        "-", "-", "-");
/* < EF */
for (i = 0; i < n; i++) {
    char temp_name[30];
    sprintf(temp_name, "%s%d %s", prefix, gb_ver, contig_name, i);
    fprintf(out,
        "CC %20.20s + %d %s %s %s\n",
        temp_name, (int)strlen(transcripts[i]), "-", "-", "-");
}

for (pass = e_rna; pass < e_done; pass++)
    for (j = 0; j < est_table_size(); j++) {
        if ((pass == e_rna && !est_table_is_rna(j)) ||
            (pass == e_est && !est_table_is_rna(j)))
            continue;
        rev = est_table_strand(j);
        fprintf(out, "CC %20.20s %c", id(j),
            (rev == 1 ? '+' : '-'));
        if (est_table_direction(j) != 0)
            fprintf(out, "%d", est_table_direction(j));
        fprintf(out, "%d/%d",
            est_table_original_len(j),
            est_table_cleaned_len(j));
        len = (dirty_tails == 'y') ?
            est_table_original_len(j) : est_table_cleaned_len(j);
        if (est_table_size() > 1)
            get_alignment_errors(hts[j], len,
                &insertions, &deletions, &mismatches, &not_aligned);
        else
            insertions = deletions = mismatches = not_aligned = 0;
        #define P(x) (100.0 * ((float)(x)) / len)
        fprintf(out, "%2f %2f",
            P(insertions), P(deletions), P(mismatches));
        #undef P
        if (est_table_seq_data(j) -> tissue)
            fprintf(out, "%14.14s ! %s\n",
                replace_char1_by_char2(est_table_seq_data(j) -> tissue,
                    ',', '.'),
                est_table_seq_data(j) -> type ?

```

io.c

```

        est_table__seq_data(j)->type : "(?)";
    else
        fprintf (out, "      i %s\n",
            est_table__seq_data(j)->type ?
            est_table__seq_data(j)->type : "(?)");
    }

    fprintf (out, "CC  \n");

    /* Print "alignments" of transcripts with consensus (doesn't compute) */
    /*EF->*/
    if (check_repeats == 'v' && (found_repeat_in_consensus & STRAIGHT_REPEAT))
        print_repeats_on_consensus(out, transcripts, transc_map, n, consensus,
            map, node_order, contig_name, "+|$", STRAIGHT_REPEAT);
    if (check_repeats == 'v' && (found_repeat_in_consensus & INVERTED_REPEAT))
        print_repeats_on_consensus(out, transcripts, transc_map, n, consensus,
            map, node_order, contig_name, "+|$", INVERTED_REPEAT);
    /*AT);
    /*<-EF*/

```

```

print_trivial_alignments(out, transcripts, transc_map, n, consensus,
    map, node_order, contig_name, "+***");

/* Print alignments of ests with consensus */
tot_ins = tot_del = tot_mis = tot_not = tot_len = 0;
for (pass = e_rna; pass <= e_rna; pass++)
    for (j = 0; j < (est_table__size() - 1); j++) {
        if (pass == e_rna && test_table__is_rna(j) ||
            pass == e_est && test_table__is_rna(j))
            continue; /* Do first rnas, then ests */

        len = (dirty_tails == 'y') ?
            est_table__original_len(j) : est_table__cleaned_len(j);
        rev = est_table__strand(j);
        printf ("%s %d ", (pass == e_rna) ? "rna" : "est", j);

        fprintf (out, "CC  to: %s from %d to %d\nCC  \n", id(j), hts[j] - y0 + 1,
            hts[j] - y0 + 1);
        fprintf (out, "CC  %s %d %s x %s %s  \n",
            prefix_gb_ver, contig_name, id(j), (rev == -1 ? "rev" : ""));
        if ((est_table__test_case(j))
            print_alignment_genweb (consensus, j, hts[j], out, map, "|$%.", '#', '***');

        else
            print_alignment_genweb (consensus, j, hts[j], out, map, "-$", '#', '***');

        if (p_est_error == 'y') {
            get_alignment_errors(hts[j], len, &insertions, &deletions,
                &mismatches, &not_aligned);

            tot_ins = insertions;
            tot_del = deletions;
            tot_mis = mismatches;
            tot_not = not_aligned;
            tot_len = len;
            printf ("error (%.2f%%, %.2f%%, %.2f%% (%d))\n",
                100 * (float) insertions / len, 100 * (float) deletions / len,

```

```

        100 * (float) mismatches / len, not_aligned);
    }

    if (est_table__size() > 1 && p_est_error == 'y') {
        est_ins = 100 * (float) tot_ins / tot_len;
        est_del = 100 * (float) tot_del / tot_len;
        est_mis = 100 * (float) tot_mis / tot_len;
        est_not = 100 * (float) tot_not / tot_len;
        printf ("total errors: (%.2f%%, %.2f%%, %.2f%% (%.2f))\n",
            est_ins, est_del, est_mis, est_not);
    }

    /* Calculate a singles count */
    for (i = a = c = g = t = o = 0; i < strlen (consensus); i++) {
        if (consensus[i] == 'A') a++;
        else if (consensus[i] == 'C') c++;
        else if (consensus[i] == 'G') g++;
        else if (consensus[i] == 'T') t++;
        else o++;
    }

    /* Print consensus sequence */
    /* SQ header line, followed by the actual sequence */
    fprintf (out, "SQ  Sequence %d BP; %d A; %d C; %d G; %d T; %d other;\n",
        a + c + g + t + o, a, c, g, t, o);
    fprintf (out, "%s\n");
    for (i = 0; i < strlen (consensus); i++) {
        fprintf (out, "%c", tolower(consensus[i]));
        if (i % 10 == 9)
            fprintf (out, " ");
        if (i % 60 == 59)
            fprintf (out, "%9d\n", i + 1);
    }
    for (; i++) {
        fprintf (out, " ");
        if (i % 10 == 9)
            fprintf (out, " ");
        if (i % 60 == 59) {
            break;
        }
    }
    /* Last line of the sequence */
    fprintf (out, "\n\n");
    if (est_table__size() > 1)
        predict_clone_length(graph -> size, cluster_name, contig_name,
            n, transc_map, map);
    print_keywords(contig_name);
    if (est_table__size() > 1)
        free(unique_path_matrix);
    fclose (out);
}

/*=====
void print_alignment_genweb(char *consensus, int est_idx, hilltop data,
FILE *fp, int *map, char *colors, char dirty_alignment_color, char poly_color)
/* print the alignment.
*/

```

```

(
    int i, x_loc, y_loc, place, svar, mode, begin_clean, end_clean;
    char dots[80], first[80], middle[80], second[80];
    char *original_seq;
    hilltop *ht;

    ht = &data;

    first[0] = middle[0] = second[0] = '\0';
    x_loc = ht->x0;
    y_loc = ht->y0;
    if (ht->type == INVERTED) mode = INVERTED;
    else mode = STRAIGHT;

    svar = (mode == INVERTED) ? -1 : 1;
    dots[50] = middle[50] = '\0';

    for (i = 0, place = 50; i < ht->len; i++) {
        if (place == 50) {
            printf (first, " %6d ", x_loc+1);
            if (ht->type == WRAPAROUND) printf (second, " ");
            else printf (second, " %6d ", y_loc+1);
            place = 0;
        }
        dots[place] = ((place % 10) == 0) ? ' ' : ' ';
        if ((first[place+9] = ht->x[i]) != '-')
            x_loc++;
        else
            first[place+9] = '-';
        if ((second[place+9] = ht->y[i]) != '-')
            y_loc += svar;
        else
            second[place+9] = '-';
        if (ht->x[i] != '-' && ht->y[i] == 'N')
            middle[place] = '-';
        if (middle[place] == '|')
            if (dirty_tails == 'y' &&
                (y_loc < est_table__first_clean(est_idx) ||
                 y_loc > est_table__first_dirty(est_idx)))
                middle[place] = dirty_alignment_color;
            else
                if (est_table__seq_data(est_idx) -> polyA_detected &&
                    y_loc+6 > est_table__original_len(est_idx))
                    middle[place] = polyA_color;
                else
                    if (est_table__seq_data(est_idx) -> polyT_detected &&
                        y_loc < 6)
                        middle[place] = colors[map[x_loc-1] % 2 + 2];
                    else
                        middle[place] = colors[map[x_loc-1] % 2];
            dots[place+1] = middle[place+1] = '\0';
        if ((place == 49) || (i == ht->len - 1)) {
            printf (&first[place + 10], " %d ", x_loc);
            if (ht->type == WRAPAROUND)
                printf (&second[place + 10], " ");
            else
                printf (&second[place + 10], " %d ", y_loc-svar+1);
            printf (fp, "CC %s\n", dots);

```

/o.c

```

    fprintf (fp, "CC %s\n", first);
    fprintf (fp, "CC %s\n", middle);
    fprintf (fp, "CC %s\n", second);
    }
    place++;
    }
    fprintf (fp, "CC \n");
}

void get_alignment_errors(hilltop *ht, int len,
    int *ins, int *del, int *mis, int *not_al)
{
    int k;

    *ins = *del = *mis = 0;
    *not_al = len;
    for(k=0; k<ht->len; k++) {
        if (ht->x[k] == ht->y[k]) continue;
        if (ht->diff[k] != '=') {
            if (ht->x[k] == '-')
                (*ins)++;
            else if (ht->y[k] == '-')
                (*del)++;
            else
                (*mis)++;
        }
        else if (ht->x[k] == '-')
            (*not_al)++;
        else
            (*not_al)++;
    }
    (*not_al) -= (ht->y0 - ht->y0 + 1);
}

/*=====
void print_alignment (hilltop data)
{
    int i, f, s, place, svar, mode;
    char dots[80], first[80], middle[80], second[80];

    first[0] = middle[0] = second[0] = '\0';
    f = data.x0;
    s = data.y0;
    if (data.type == INVERTED) mode = INVERTED;
    else mode = STRAIGHT;

    svar = (mode == INVERTED) ? -1 : 1;
    dots[50] = middle[50] = '\0';

    for (i = 0, place = 50; i < data.len; i++) {
        if (place == 50) {
            printf (first, " %6d ", f+1);
            if (data.type == WRAPAROUND) printf (second, " ");
            else printf (second, " %6d ", s+1);
            place = 0;
        }
        dots[place] = ((place % 10) == 0) ? ' ' : ' ';
        if ((first[place+9] = data.x[i]) != '-') f++;
        if ((second[place+9] = data.y[i]) != '-') s += svar;
        middle[place] = (data.diff[i] == '=') ? ' ' : data.diff[i];
        dots[place+1] = middle[place+1] = '\0';
    }
}
=====*/

```

/o.c


```

if ((place == 49) || (i == data.len - 1)) {
    printf (&first[place + 10], " %d ", f);
    if (data.type == WRAPAROUND)
        printf (&second[place + 10], " ");
    else
        printf (&second[place + 10], " %d ", s-svar+1);
    printf (" %s\n", dots);
    printf ("%s\n", first);
    printf ("%s\n", middle);
    printf ("%s\n", second);
}
}
place++;
}
printf ("\n");
}

void print_trivial_alignments(FILE *out, char **transcripts, int **transc_map,
int n, char *consensus, int *map, int *node_order,
char *contig_name, char *colors)
{

```

```

    int i;
    char upper[80], middle[80], lower[80], upper_prefix[10], lower_prefix[10];
    int cons_pos, line_pos, trans_pos, start, node_pos;
    int cons_len, trans_len;
    char *dots = "          ";

    cons_len = strlen(consensus);
    for (i = 0; i < n; i++) {
        printf (out, "CC to: %s%d %s %d from %d to %d\nCC \n",
            prefix, gb_ver, contig_name, i,
            1, (int)strlen(transcripts[i]));
        printf (out, "CC %s%d %s x %s%d %s %d ..\nCC \n",
            prefix, gb_ver, contig_name,
            prefix, gb_ver, contig_name, i);

        /** print the alignment **/
        start = trans_pos = line_pos = cons_pos = node_pos = 0;
        trans_len = strlen(transcripts[i]);
        while (cons_pos < cons_len && trans_pos < trans_len) {
            if (node_order[map[cons_pos]] == transc_map[i][node_pos]) {
                start = i;
                if (line_pos == 0) {
                    printf(upper_prefix, " %6d ", cons_pos+1);
                    printf(lower_prefix, " %6d ", trans_pos+1);
                }
                upper[line_pos] = lower[line_pos] = consensus[cons_pos];
                middle[line_pos] = colors[map[cons_pos] % strlen(colors)];
                line_pos++;
                cons_pos++;
                trans_pos++;
                if (cons_pos < cons_len && map[cons_pos] != map[cons_pos-1])
                    node_pos++;
            }
            else {
                if (!start) {
                    cons_pos++;
                    continue;
                }
            }
        }
    }
}

```

```

if (line_pos == 0) {
    printf(upper_prefix, " %6d ", cons_pos+1);
    printf(lower_prefix, " %6d ", trans_pos+1);
}
upper[line_pos] = consensus[cons_pos];
middle[line_pos] = 32;
lower[line_pos] = '-';
line_pos++;
cons_pos++;
}
if (line_pos >= 50) {
    /** print the data **/
    printf (out, "CC %s %s %d\n", %s\n", line_pos, dots);
    printf (out, "CC %s %s %d\n", upper_prefix, line_pos, upper, cons_pos);
    printf (out, "CC %s %s %d\n", upper_prefix, line_pos, upper, cons_pos);
    printf (out, "CC %s %s %d\n", lower_prefix, line_pos, lower, trans_pos);
    line_pos = 0;
}
}
if (line_pos > 0) {
    /** print the rest of the data **/
    printf (out, "CC %s %s %d\n", %s\n", line_pos, dots);
    printf (out, "CC %s %s %d\n", upper_prefix, line_pos, upper, cons_pos);
    printf (out, "CC %s %s %d\n", upper_prefix, line_pos, upper, cons_pos);
    printf (out, "CC %s %s %d\n", lower_prefix, line_pos, lower, trans_pos);
    line_pos = 0;
}
}
printf(out, "CC \n");
}
}

char *replace_char1_by_char2(char *s, char char1, char char2)
{
    char *tmp;
    for(tmp = s; *tmp; tmp++)
        if (*tmp == char1)
            *tmp = char2;
    return s;
}

/* ----- Routines to print each tissue *once* for each transcript ----- */

tissue_hash new_tissue_hash(void)
{
    int i;
    tissue_hash t;

    if ((t = malloc(sizeof(*t))) == NULL)
        return NULL;

    for(i=0; i<TBL_SZ; i++) {
        (*t)[i].name = NULL;
        (*t)[i].multiplicity = 0;
    }
    return t;
}

```

```

    }
    static int tissue_hash_prim(char *s)
    {
        int i;
        unsigned int hash = 0x29061971; /* My birthday! */
        for(i=0; s[i]; i++)
            hash = (hash >> 14) ^ ((hash ^ (s[i]*127)) & 0xffff) << 16);
        return hash % TBL_SZ;
    }

    static int tissue_hash_sec(int key)
    {
        return key + 17;
    }

    int tissue_hash_intern(tissue_hash_t, char *s)
    {
        int i, key;
        key = tissue_hash_prim(s);
        i = TBL_SZ;
        do {
            if ((t)[key].name == NULL) {
                /* Not found */
                (t)[key].name = s;
                (t)[key].multiplicity = 1;
                return 0;
            }
            else if (strcmp((t)[key].name, s) == 0) {
                /* Found */
                (t)[key].multiplicity++;
                return 1;
            }
            key = tissue_hash_sec(key);
            i--;
        } while (i > 0);
        err("tissue hash table FULL (key='%s', primary hash = %d)",
            s, tissue_hash_prim(s));
        return -1;
    } /* UNREACHED */

    void tissue_hash_print_on_string(tissue_hash_t, char *str, char *pre, char *sep)
    {
        int i;
        short first;
        char tmp[500];
        for(i=0; first=1; i<TBL_SZ; i++)
            if ((t)[i].name) {
                strcpy(tmp, (t)[i].name);
                replace_char1_by_char2(tmp, '_', ' ');
                if (first) {
                    sprintf(str, "%s %s(%d)", str, pre, tmp, (t)[i].multiplicity);
                    first = 0;
                }
            }
    }

```

```

    }
    else
        sprintf(str, "%s%s(%d)", str, sep, tmp, (t)[i].multiplicity);
    }

    static void assembly_type_report(FILE *fp, splice_graph *graph,
        int cluster_volume, int consensus_len,
        int unique_order)
    {
        int cycles = splice_graph->is_undirected_cycles(graph);

        if (graph->size > 1) {
            fprintf(fp, "CC Multiple transcripts, "
                "putative alternative splicing\n");
        }
        /* Now part of the general record/report_warn() facility! */
        if (test_table_any_ximeric())
            fprintf(fp, "CC Warning : possible chimeric sequence; Chimeric transcript
                not shown\n");
        #endif
        if (cycles) {
            fprintf(fp, "CC inner spliced-out segments were detected\n");
            if (splice_graph_num_initial(graph) == 1)
                fprintf(fp, "CC its 5' end is common to all transcripts\n");
            if (splice_graph_num_terminal(graph) == 1)
                fprintf(fp, "CC its 3' end is common to all transcripts\n");
            fprintf(fp, "CC Segment order in generalized transcript is% determined\n",
                (unique_order)?":":" not");
        }
        fprintf(fp,
            "CC There are %d nucleotides in the sequences of the cluster;\n"
            "CC Reduced to %d in the generalized transcript.\n",
            cluster_volume, consensus_len);
        if (found_repeat_in_consensus & STRAIGHT_REPEAT)
            fprintf(fp, "CC Cluster is suspected to contain repeats.\n");
        fprintf(fp, "CC Technicalities: %d nodes %d-> %d %d %d\n",
            graph->size,
            splice_graph_num_initial(graph),
            splice_graph_num_terminal(graph),
            cycles,
            (splice_graph_num_initial(graph) +
             splice_graph_num_terminal(graph) == 2)?
            "(very-interesting)": "(interesting)");
    }

    /* get all needed categories for specifying the cluster. Note, consensus_len
    and unique_order should be (have already been) determined elsewhere
    RETURNS static pointer on success (graph is connected), NULL otherwise
    */
    cluster_classification *get_cluster_classification(splice_graph *graph)
    {
        static cluster_classification clstr_class;
        clstr_class.num_initial_nodes = splice_graph_num_initial();
        clstr_class.num_terminal_nodes = splice_graph_num_terminal();
    }

```

```

if ((clstr_class.undirected_cycles =
    splice_graph__is_undirected_cycles(graph)) == -1)
    return NULL;
    clstr_class.distinguished_ends =
    (clstr_class.num_initial_nodes == 1 &&
     clstr_class.num_terminal_nodes == 1);
    clstr_class.ximetric = est_table__any_ximetric();
    clstr_class.volume = get_cluster_volume();
    return &clstr_class;
}

void splicemarks_print(char *str, int *map, splice_graph *graph)
{
    int i, len;

    len = 0;
    for(i=0; map[i+1] != -1; i++) {
        len += strlen(graph->node_list[map[i]]->seq);
        sprintf(str, "%s %p %d <td>td>", str, len,
            graph->node_list[map[i]]->out_degree,
            graph->node_list[map[i+1]]->in_degree);
    }
}

/*=====
*/
/* * post-processing involving in singleton clusters
*/

void deal_with_singletons(FILE *cview_file, char *contig_name)
{
    char *transcript, *tmp;
    int *transc_map, consensus_len, *node_order, *map;
    char *tr_header=NULL;
    tissue_hash tissue[1];
    extern FILE *graph_file;
    extern char cluster_name[];
    extern FILE *transc_file;

    print_title();
    consensus_len=strlen(est_table__original_seq(0));
    transcript = strdup(est_table__original_seq(0));
    map = (int *) calloc(consensus_len, sizeof(int));
    /* memset(map, 0, consensus_len+1); something goes wrong in SUN here */
    transc_map = (int *) malloc(sizeof(int)*2);
    node_order = (int *) malloc(sizeof(int)*2);
    transc_map[0] = 0;
    node_order[0] = 0;
    transc_map[1] = -1;
    node_order[1] = -1;
    tissue[0] = new_tissue_hash();

    if (post_process=='y') {
        write_cluster_output(&transcript, 1, transcript,
            consensus_len, node_order,
            1,
            contig_name, map,

```

i.o.c

```

    &transc_map, tissue,
    NULL,
    NULL);
}
if (graph_file) {
    cprof p;
    fprintf(graph_file, "CONTIG %s.%s\n", cluster_name, contig_name);
    /* DOC fields go here */
    fprintf(graph_file, "NODES 1\n");
    /* No edges */
    fprintf(graph_file, "NODE 0\n");
    /* End of (dummy) record */
    fprintf(graph_file, "ENDCONTIG\n");
}

if (cview_file || transc_file) {
    write_transcripts_headers(&tr_header, contig_name, cluster_name,
        &transc_map, &transcript, 1,
        tissue, NULL);
}

if (transc_file) {
    write_transcripts(transc_file, &tr_header, &transcript, 1);
}

if (cview_file) {
    cv_write_contig(cview_file, contig_name, est_table__original_seq(0), 1,
        map, &tr_header, &transc_map, node_order, NULL);
}

free(transcript);
free(map);
free(transc_map);
free(node_order);
free(tissue[0]);
if (tr_header) free(tr_header);
}
/*=====
*/
/* * Sequence the activities needed to produce output to the EMBL and
* * transcript files.
*/

void cluster_post_process(splice_graph *graph,
    char *contig_name, char *cluster_name)
{
    char consensus[MAX_TRANSC_LEN]; /* agreed sequence */
    int map[MAX_TRANSC_LEN]; /* map of consensus positions to nodes */
    int node_order_in_consensus[MAX_NODES+1];

    char *transcripts[MAX_TRANSC+1]; /* possible transcripts */
    int *transc_map[MAX_TRANSC+1]; /* map of transcript positions to nodes */
    char *tr_headers[MAX_TRANSC+1];
    int *transc_est_list[MAX_TRANSC]; /* ests agreeing with transcripts */

    tissue_hash tissue[MAX_TRANSC]; /* tissues for each transcript */

    extern char splicemarks_p_graph_cprofs;

```

i.o.c

```

extern int directed_cycle;

extern FILE *cview_file, *transc_file, *graph_file;

int tot_err, tot_cor;
float err_cor;

int i, node_idx, num_transcripts = 0, repeat_detected;
hilltop **hts;
short unique_order;
int num_ximeric_edges;

int *components; /*Array which hold in the i entry the connective
                  component of the i'th node */
int comp_num; /*Hold the number of connective components */
int mark_ximeric(splice_graph *);

/*
 * Mark everything we allocate later as unused (NULL) so that we can
 * tell if we should free it at the end. This is important if we
 * encounter an error!
 */
hts = NULL;
for(i=0; i<MAX_TRANSC; i++) {
    transcripts[i] = NULL;
    transc_map[i] = NULL;
    tr_headers[i] = NULL;
}
transc_map[MAX_TRANSC] = NULL;

tot_err = tot_cor = 0;

if(p_graph.cprofs == 'y')
    graph__print_cprofs(graph, stdout);

/* for each est finds its path and put it in the field hyper_edge of est*/
/* Note: test node also does so, we will remove it soon */
find_est_paths(graph);
print_ests_hyper_edge_alignment();

/* Compute sequences held by each node */
for(node_idx=0; node_idx < graph->size; node_idx++) {
    graph->node_list[node_idx]->marked = 0;
    graph->node_list[node_idx]->seq =
        cprof_compute_assembly(graph->node_list[node_idx]->profile, 1,
                               &graph->node_list[node_idx]->err_num,
                               &graph->node_list[node_idx]->corrected_err);
}

if (!splice_graph__is_connected(graph)) {
    if (components = malloc(graph->size*sizeof(int)) == NULL)
        err("memory failure in cluster_post_process (%d bytes)\n",
            graph->size*sizeof(int));
    /* The graph is disconnected - we find the connected components */
    /* in the future we will re run for each component */
}

```

```

comp_num = splice_graph__find_connective_components(graph, components);
mark_ests_components(graph, components);
print_connective_components(graph, components, comp_num);
free(components);
record_err(DISCONNECTED_MSG);
}

if (!err())
    if(rp_mode != 'y' && rp_mode != 't' && rp_mode != 's' || !need_rpmode) {
        if (!build_transcripts(graph, transcripts, &num_transcripts, transc_map)) {
            record_err(DIRECTED_CYCLE_MSG);
            directed_cycle = 1;
        }
    }
    else {
        rp_mode_build_transcripts(graph, transcripts, &num_transcripts,
                                transc_map, &directed_cycle);
        directed_cycle = find_cycle(graph, 0); /* 0 for not printing the cycles */
        if(directed_cycle) {
            for(i=0; i<num_transcripts && i<MAX_TRANSC; i++)
                if ((tissue[i] = new_tissue_hash()) == NULL)
                    err("memory failure in cluster_post_process: "
                        "failed to allocate tissue hash #&d", i);
            /* tissue_hash_fill looks at node_path of ests,
             * therefor we have to put hyper_edge in the node path place */
            est_table_switch_node_path_hyper_edge();
            tissue_hash_fill(tissue, transc_map, num_transcripts);
            est_table_switch_node_path_hyper_edge();

            /* Write transc_file */
            if (transc_file) {
                write_transcripts_headers(tr_headers, contig_name, cluster_name,
                                        transc_map, transcripts, num_transcripts,
                                        tissue, graph);
                write_transcripts(transc_file, tr_headers, transcripts, num_transcripts);
            }

            for(i=0; i<num_transcripts && i<MAX_TRANSC; i++)
                free(tissue[i]);

            for(i=0; i<MAX_TRANSC; i++) {
                if (transc_map[i])
                    free(transc_map[i]);
                if (transcripts[i])
                    free(transcripts[i]);
                if (tr_headers[i])
                    free(tr_headers[i]);
            }
            return;
        }
        /* Create a consensus sequence, incidentally checking for cyclic graph */
        if (!err()) {
            if ((unique_order = build_consensus(graph, consensus,
                                                map, node_order_in_consensus)) == -1) {
                record_err(DIRECTED_CYCLE_MSG);
                directed_cycle = 1;
            }
        }
    }
}

```

```

if (! erred()) {
    /* Diagnose problems with consensus (repeats are tested in
       write_cluster_output()...)
       if (strcmp(cmp_res,"none") != 0)
           compare_consensus(consensus);

    /* Align all ests against the consensus */
    hts = get_est_alignments(consensus);
}

if (! erred())
    for(i=0; i<est_table__size(); i++) {
        /* Set nodes est i goes through */
        test_nodes(*hts[i], map, i, graph, node_order_in_consensus);
        /* compare the test_node path with the hyper_edge */
        if(!est_table__verify_est_path(i)) {
            record_warn("test node and hyper_edge disagree on est %d path\n", i);
        }
    }

if (! erred()) {
    /* Mark chimeric edges as such, so they're not included in transcripts */
    num_ximERIC_edges = mark_ximERIC(graph);

    if (num_ximERIC_edges)
        record_warn("possible chimeric sequence; Chimeric transcript not shown");
}

for(i=0; i<num_transcripts && i<MAX_TRANSC; i++)
    if (! (tissue[i] = new_tissue_hash()) == NULL)
        err("memory failure in cluster post process: "
            "failed to allocate tissue hash #&d", i);

if (! erred())
    if (p_error_data == 'y') {
        printf("Error correction data:\n");
        for(i=0; i<graph->size;i++) {
            splice_node *node = graph->node_list[i];

            printf("node %d: length %d errors %d corrected %d \n", i,
                splice_node__len(node), node->err_num, node->corrected_err);
            tot_err+=node->err_num;
            tot_cor+=node->corrected_err;
        }
        err_cor = 100*(float)tot_cor/tot_err;
    }

/* check_is_undirected_cycles?? */

if (! erred()) {
    /*
     * Write EMBL output (if looking just for "interesting", find a cycle
     * before printing
     */
    if (post_process != 'i' ||
        splice_graph__is_undirected_cycles(graph)!=1)
        write_cluster_output(transcripts, num_transcripts,

```

```

consensus, strlen(consensus),
node_order_in_consensus,
unique_order,
contig_name_map,
transc_map,
tissue, graph, hts);

/* output for cluster_view, if needed */
if (graph_file)
    graph_write_record(graph_file, graph, cluster_name, contig_name,
        num_transcripts, transc_map);

/* create transcripts header */
if (cview_file || transc_file) {
    write_transcripts_headers(tr_headers, contig_name, cluster_name,
        transc_map, transcripts, num_transcripts,
        tissue, graph);
}

/* Write transc file */
if(transc_file) {
    write_transcripts(transc_file, tr_headers, transcripts, num_transcripts);
}

if (cview_file) {
    cv_write_contig(cview_file, contig_name, consensus, num_transcripts,
        map, tr_headers,
        transc_map, node_order_in_consensus, hts);
}

/* Clean up -- go here even after errors */
for(i=0; i<num_transcripts && i<MAX_TRANSC; i++)
    free(tissue[i]);

for(i=0; i<MAX_TRANSC; i++) {
    if (transc_map[i])
        free(transc_map[i]);
    if (transcripts[i])
        free(transcripts[i]);
    if (tr_headers[i])
        free(tr_headers[i]);
    free_est_alignments(hts);
}

/* mark all edges suspected to be ximERIC and all ests supporting those edges.
   normally we expect at most one such edge (and the ests are of one clone)
*/
int mark_ximERIC(splice_graph *graph)
{
    int source, target, ximERIC_count=0;

    if (graph && graph->size>3)
        for (source=0; source<graph->size; source++)
            for (target=0; target<graph->size; target++)

```

```

if (source==target) continue;
if (splice_graph___is_ximeric_edge(graph,source,target))/*EF
{
    splice_graph___set_edge_and_ests_ximeric(graph,source,target);/*EF
    ximeric_count++;
}
return ximeric_count;
}

/*EF-->*/
void print_repeats_on_consensus(FILE *out, char **transcripts,
int **transc_map,int n,
char *consensus,int *map,int *node_order,
char *contig_name,char *colors, int type)
{
    int i;
    char upper[80],middle[80],lower[80],upper_prefix[10],lower_prefix[10];
    int cons_pos,line_pos,trans_pos,start,node_pos;
    int cons_len,trans_len;
    char *dots = "
";
    char *inv,*tmp;
    hilltop *list,*ht;
    int in_repeat = 0;

    cons_len = strlen(consensus);
    printf("writing repeats on consensus ...");
    if (type == STRAIGHT_REPEAT)
        list = Hilltops (consensus,consensus, STRAIGHT, cutoff, 2*bound);
    else if (type == INVERTED_REPEAT) {
        list = Hilltops (consensus, inv = subseq(consensus, cons_len, 0),
        INVERTED, cutoff, 2*bound);
        free(inv);
    }
    else
        err("Unknown consensus repeats type %d", type);
    for (ht = list; ht != NULL; ht = ht->next) {
        get_alignment(consensus, consensus, ht, BEST);
        if (type == INVERTED_REPEAT) {
            int ytmp = cons_len-1-ht->yt;
            ht->yt = cons_len-1-ht->y0;
            ht->y0 = ytmp;
        }
        fprintf (out,"CC to: %s%s from %d to %d\nCC \n",
            prefix, gb_ver,contig_name,
            1, cons_len);
        fprintf (out,"CC %s%s x %s%s \nCC \n",
            prefix,gb_ver, contig_name,

```

```

prefix,gb_ver, contig_name);

line_pos=0;
cons_pos=0;
while (cons_pos<cons_len) {
    upper[line_pos] = lower[line_pos] = consensus(cons_pos);
    in_repeat = 0;
    for (ht = list; ht != NULL; ht = ht->next)
        {
            if (abs(ht->x0-ht->y0) > 25)
                {
                    if ((cons_pos >= ht->y0 && cons_pos <= ht->yt) &&
                        (cons_pos >= ht->x0 && cons_pos <= ht->xt) )
                        {
                            middle[line_pos] = colors[3];
                            in_repeat = 1;
                            continue;
                        }
                    if (cons_pos >= ht->y0 && cons_pos <= ht->yt)
                        {
                            middle[line_pos] = colors[1];
                            in_repeat = 1;
                            continue;
                        }
                    if (cons_pos >= ht->x0 && cons_pos <= ht->xt)
                        {
                            middle[line_pos] = colors[2];
                            in_repeat = 1;
                            continue;
                        }
                }
            if (!in_repeat)
                middle[line_pos] = colors[0];
            if (line_pos == 0) {
                printf(upper_prefix," %d ",cons_pos+1);
                printf(lower_prefix," %d ",cons_pos+1);
            }
            line_pos++;
            cons_pos++;
            if (line_pos>=50) {
                /** print the data **/
                printf (out,"CC %s%s %s\n",upper_prefix,line_pos, upper,cons_pos);
                printf (out,"CC %s%s %s\n",line_pos,middle);
                printf (out,"CC %s%s %s\n",lower_prefix,line_pos, lower,cons_pos);
                line_pos = 0;
            }
        }
    if (line_pos > 0) {
        /** print the rest of the data **/
        printf (out,"CC %s%s %s\n",line_pos, dots);
        printf (out,"CC %s%s %s\n",upper_prefix,line_pos, upper,cons_pos);
        printf (out,"CC %s%s %s\n",line_pos,middle);
        printf (out,"CC %s%s %s\n",lower_prefix,line_pos, lower,cons_pos);
    }
}

```

```

    fprintf(out, "CC %s %s %d\n", lower_prefix, line_pos, lower_cons_pos);
}

fprintf(out, "CC \n");
destroy_hilltop_list(&list);
}

/*=====
**
** for simulations. compare seq to the (first) sequence if the fasta file
** <cmp_res>
**
static void compare_consensus(char *seq)
{
    fasta_file ff;
    rich_fasta_seq_ptr rfp;
    hilltop *ht;
    int insertions, deletions, mismatches, unaligned;
    int k;

    if(!((ff = rich_fasta_open(cmp_res))) {
        printf("Can't open compare file. ignoring...\n");
        return;
    }
}

```

```

while(rfp = rich_fasta_read_seq(ff, ASSEMBLY_MODE)) {
    printf("Aligning %s with consensus\n", rfp->name);
    insertions = deletions = mismatches = 0;
    unaligned = rfp->original_len;
    ht = get_six18_alignment(rfp->original_seq, seq,
                             SEPARATE_HILLTOPS|OVERLAP_MODE);
    print_alignment(*ht);
    printf("calculating errors ....\n");
    for(k=0; k<ht->len; k++) {
        if (ht->x[k] == ht->y[k]) continue;
        if (ht->x[k] == '-' || ht->y[k] == '-') {
            insertions++;
            continue;
        }
        if (ht->y[k] == '-') {
            deletions++;
            continue;
        }
        mismatches++;
    }
    unaligned -= (ht->y0 - ht->y0 + 1);
    printf("Errors in comparison: ins %2f%% del %2f%% mis %2f%% unaligned %d\n",
          100*(float)insertions/rfp->original_len,
          100*(float)deletions/rfp->original_len,
          100*(float)mismatches/rfp->original_len, unaligned);

    rich_fasta_seq_free(rfp);
}
rich_fasta_close(ff);
}

```

```

/* * Print the predicted clone length and the length along each transcript
*/
static void predict_clone_length(int graph_size, char *cu_name, char *cn_name,
                                int n_transc, int *transc_map, int *cons_map)
{
    int five_p, three_p, i, j, clength, t;

    for (i=0; i<est_table__clone_size(); i++)
        if ((j = est_table__clone_mate(i)) > i &&
            est_table__strand(i) != est_table__strand(j)) {
            five_p = est_table__strand(i)==1?i:j;
            three_p = est_table__strand(i)==-1?i:j;
            if ((clength = (est_table__clone_length(five_p, three_p))<=0)
                continue;
            printf("#GS %d #CU %s #CN %s #CL %s $s -----> $s assembly %d\n",
                  "genbank %d\n",
                  graph_size, cu_name, cn_name,
                  est_table__clone_name(i),
                  id(five_p), id(three_p),
                  est_table__get_end_in_cons(three_p) -
                  est_table__get_start_in_cons(five_p) +
                  est_table__trim5(five_p) + est_table__trim3(three_p),
                  est_table__clone_length(five_p, three_p));
        }
}

```

```

void get_transcript_alignment(FILE *fp, char *consensus, int *map,
                             int *transc_map, int *node_order)
{
    int cons_pos, line_pos, transc_pos, start, transc_pos_in, cons_pos_in, node_pos;
    int cons_len, transc_len, in;

    cons_len = strlen(consensus);
    in = transc_pos = cons_pos = transc_pos_in = cons_pos_in = node_pos = 0;
    while (cons_pos<cons_len) {
        if (node_order[map[cons_pos]] == transc_map[node_pos]) {
            if (in==0) {
                in = 1;
                cons_pos_in = cons_pos;
                transc_pos_in = transc_pos;
            }
            cons_pos++;
            transc_pos++;
            if (cons_pos < cons_len && map[cons_pos] != map[cons_pos-1])
                node_pos++;
        }
        else {
            if (in==1) {
                in = 0;
                fprintf(fp, "ALIGN (%d,%d)x(%d,%d)\n", cons_pos_in, cons_pos-1,
                        transc_pos_in, transc_pos-1);
                cons_pos++;
            }
            if (in==1) {
                fprintf(fp, "ALIGN (%d,%d)x(%d,%d)\n", cons_pos_in, cons_pos-1,

```



```

    )
    trans_pos_in.trans_pos-1);
}

void write_transcript_header(char *header, char *contig_name, char *cluster_name,
    int *tr_map, tissue_hash tissue, splice_graph *graph,
    int tr_len, int idx)
{
    int node_idx;
    extern char splicemarks;

    printf(header, "%s %s %d %d" header, prefix, gb_ver, contig_name, idx);
    if (gb_ver >= 104) {
        if (cluster_name[0])
            printf(header, "%s #CU %s" header, cluster_name);
        if (contig_name[0]) /* should *always* be present, really! */
            printf(header, "%s #CN %s" header, contig_name);
    }
    printf(header, "%s #TY TRS" header);

    printf(header, "%s #ES %d #RN %d" header,
        est_table__est_num(),
        est_table__rna_num());
    /* Transcript path through graph */
    printf(header, "%s #PH" header); /* changed from PT */
    for (node_idx = 0; tr_map[node_idx] != -1; node_idx++)
        printf(header, "%s %d" header,
            node_idx, tr_map[node_idx]);

    /* Simple headers */
    printf(header, "%s #DT %s #LN %d" header, date, tr_len);
    /* Write all tissues contributing to this transcript */
    tissue_hash_print_on_string(tissue, header, " #TI", " ");
    if (graph != NULL && splicemarks == 'Y') /* Annotate transcript's splice po
ints */
        splicemarks_print(header, tr_map, graph);

    strcat(header, "\n");
}

void write_transcripts_headers(char **headers, char *contig_name, char *cluster_n
ame,
    int **tr_map, char **transcripts, int num_transcrip
ts,
    tissue_hash *tissue, splice_graph *graph)
{
    int i;
    char *tmp;

    for (i=0; i<num_transcripts && i<MAX_TRANSC; i++) {
        tmp = transcripts[i];
        if ((headers[i] = (char *)malloc
            (MAX_HEADER * sizeof (char))) == NULL)
            err("memory failure allocating for transcript_header");
        headers[i][0] = '\0';
        write_transcript_header(headers[i], contig_name, cluster_name, tr_map[i],
            tissue[i], graph, strlen(tmp), i);
    }
}

```

Io.C

```

    )
    void write_transcripts(FILE *fp, char **tr_headers, char **transcripts,
        int num_transcripts)
    {
        int i;
        char *tmp;

        for (i=0; i<num_transcripts && i<MAX_TRANSC; i++) {
            char *tmp = transcripts[i];
            fprintf(fp, "%s", tr_headers[i]);
            while (strlen(tmp)>60) {
                tmp+=60;
                fprintf(fp, "%s\n", tmp);
            }
        }

        void tissue_hash_fill(tissue_hash *tis, int **transc_map, int transc_number)
        {
            int est_no, transc_no, first, last;

            for (est_no = 0; est_no < est_table__size(); est_no++) {
                for (transc_no = 0; transc_no < transc_number; transc_no++) {
                    if (!get_est_bounds_in_transcript(est_no, transc_map(transc_no),
                        &first, &last))
                        continue;

                    if (tis && est_table__seq_data(est_no) -> tissue)
                        (void)tissue_hash_intern(tis[transc_no],
                            est_table__seq_data(est_no) -> tissue);
                }
            }

            void mark_ests_components(splice_graph *graph, int *components)
            {
                int i, j, est_num;
                identifier_t node_ests[MAX_CLUSTER_SIZE];
                for (i=0; i<graph->size; i++) {
                    est_num = cprof_ids(graph->node_list[i] -> profile, node_ests,
                        MAX_CLUSTER_SIZE);
                    for (j=0; j<est_num; j++) {
                        est_table__set_component(node_ests[j], components[i]);
                    }
                }

                void print_connective_components(splice_graph *graph, int *components,
                    int comp_num)
                {
                    int comp_idx, i;
                    for (comp_idx=0; comp_idx<comp_num; comp_idx++) {

```

Io.C

```
printf("connective_component %d:\n",comp_idx);
printf("Ests:\n");
for(i=0;i<est_table___size();i++) {
    if(est_table___get_component(i)==comp_idx) {
        printf(" %d %s\n",i,est_table___seq_data(i)->header);
    }
}
printf("\n");
printf("Nodes:\n");
for(i=0;i<graph->size;i++) {
    if(components[i]==comp_idx) {
        printf(" %d",i);
    }
    printf("\n");
}

void print_ests_hyper_edge_alignment()
{
    int i,j;
    hyper_edge_node *he_nodes,*he;
    printf("\nEsts Aligmet:\n\n");
    for(i=0;i<est_table___size();i++) {
        printf("Est %d\n",i);
        he_nodes = est_table___get_hyper_edge(i);
        for(j=0,he=he_nodes; he->node_id != -1; j++,he=he_nodes[j]) {
            printf("node%d <-> est%d %d <-> %d.%d\n",he->node_id,i,
                he->cprof_start,he->cprof_end,he->est_start,he->est_end);
        }
        printf("\n");
    }
}
```



```
/* $Log: main.c,v $
* Revision 1.54 1998/06/30 22:21:14 avner
* rp_mode = s
*
* Revision 1.53 1998/06/30 22:15:16 avner
* remove parameters 'iter_gap_major', 'final_gap_major' which belong to the old
* multiple alignment engine.
* minor changes in parameters for gbl07.
*
* Revision 1.52 1998/06/25 08:54:46 ariels
* Add "maximal cprof memory" parameters giving maximal values of
* maximal cprof memory.
*
* Revision 1.51 1998/06/24 14:02:31 ariels
* Determine memory use of alignment (quadratic-space limit and
* log-likelihood paging table) according to machine specifics.
*
* Revision 1.50 1998/06/24 07:18:48 eyal
* Add parameter node_path_mode.
*
* Revision 1.49 1998/06/18 14:42:47 ariels
* Strip "#ifdef CPROF_ALIGN" and "#ifdef IMPROVE_CPROF" lines.
*
* Revision 1.48 1998/05/15 10:53:30 avner
* changing rna_test mode to dont_assembly, a parameter with few modes. It
* defines which sequence should be viewed in the final comparison, and NOT be
* assembled. It can be all rnas (good for simulations), the biggest rna, or the
* seq of sequences whose name starts with a given prefix (good for wetlab).
*
* Revision 1.47 1998/05/11 14:42:20 eyal
* Add parameter p_graph_cprofs.
*
* Revision 1.46 1998/04/25 18:04:13 eyal
* Change defaults rp_mode=n with_rna=y
*
* Revision 1.45 1998/04/24 12:46:53 avner
* 'rich_fasta_read_seq' is called with the ASSEMBLY_MODE parameter.
*
* Revision 1.44 1998/04/24 09:34:54 avner
* change parametrers for the 905 run.
*
* Revision 1.43 1998/04/23 15:22:54 ariels
* Fix warned to warned() (need a function call to check if any warning
* occurred, not a function ptr!)
*
* Revision 1.42 1998/04/22 12:41:25 ariels
* Add "flip_interactively" option for testing flipper (allows you to specify
* for each sequence if you wish to flip it).
*
* Make stdout line-buffered even when going to a file.
*
* Revision 1.41 1998/04/18 22:06:53 avner
* change use pf 'dirty_tails' It is now a y/n question.
* change some parameters. Most important the dirty probability parameters (it
* is mostly big insertion rate that makes the difference).
*
* Revision 1.40 1998/04/15 15:08:45 ariels
```

main.c

```
* Add clean_(change,del,ins) parameters for dealing with "very clean"
* sequences in the multiple alignment.
*
* Revision 1.39 1998/04/15 09:30:48 ariels
* Read parameters for probabilities of mismatch/insertion/deletion at
* "dirty" locations.
*
* Revision 1.38 1998/04/13 07:24:13 eyal
* 1. Replace the parameter transc_db to transc_file
* 2. open the transc_file in function prologue, and use global var transc_file
* instead of transc_bd in all other files
*
* Revision 1.37 1998/04/09 13:48:34 eyal
* Take care of correct printing in assembling clustrs
*
* Revision 1.36 1998/04/09 13:16:54 eyal
* Add treatment for assembling clusters (assembly_unit==u)
*
* Revision 1.35 1998/04/08 06:42:04 eyal
* Add Skip RNA feature
*
* Revision 1.34 1998/04/06 08:41:03 avner
* change default for dir and in. add printing of command-line.
*
* Revision 1.33 1998/03/29 20:49:03 avner
* add flag 'dirty_tails'. When 'a' we assemble witht the whole sequence, when
* 'i' only with the cleaned, and when 's' also assemble with the cleaned, but
* show the tails in the post-process alignment.
*
* Revision 1.32 1998/02/22 22:04:56 avner
* get rid of parameters 'p_est_align' who wasn't used, and 'throw_gaps' who was
* always set to 'y' conceptually, and so became obsolete.
*
* Revision 1.31 1998/02/22 17:15:20 ariels
* Use new error and warning reporting facility.
*
* Revision 1.30 1998/02/21 23:48:54 avner
* Changing defaults: 'indep_node_len' 15->20.
* Lot's of changes in 'print_contig_online_report':
* make a better distinction between straight and inverted repeats.
* print the similarity of the best repeat of each kind.
* Give better output to cyclic graphs.
*
* Revision 1.29 1998/02/19 09:25:27 ariels
* Add graph_file program argument and external variable, to write graph
* structure file.
*
* Revision 1.28 1998/02/17 16:13:26 ariels
* Keep track of cluster name (as well as contig name).
* Fix name of global variable cluster_no (to contig_name), since it was
* neither CLUSTER nor NO (number)...
*
* Revision 1.27 1998/02/15 22:30:54 avner.
* change 'cluster' to 'contig' in 'print_contig_online_report' (name of funct
* ion
* also changed).
* After experimenting, return value of cprof_lgapop to 45 (from 24).
*
* Revision 1.26 1998/02/14 17:35:58 avner
* Change 'print_title' to be nice.
```

main.c

```
* Revision 1.25 1998/02/10 11:13:32 avner
* Remove variable 'olap_fuzziness' (indep_node_len will serve instead).
*
* Revision 1.24 1998/02/09 06:54:32 avner
* (For flipper) dump many parameters. Degenerate the building of olap_graph,
* the coloring and inverting parts. This is why we don't need pairs file no
* longer.
*
* Revision 1.23 1998/02/06 14:14:47 avner
* Changing default parameters: cprof_lgapop (from 48 to 24) and other print-par
* ameters.
*
* Revision 1.22 1998/02/01 06:53:19 eval
* Adding the parameters phase1_id_bound and phase2_id_bound.
*
* Revision 1.21 1998/01/21 07:53:52 ariels
* Put err() in scope of prototype (add #include "error.h")
*
* Revision 1.20 1998/01/14 10:05:54 eval
* Add support for rna_test_mode.
*
* Revision 1.19 1998/01/13 09:49:04 ariels
* Commented-out P_ERROR flag. Broken prm_argv() "always" prints "unused
* arguments" error message (and aborts silently), even if all args are
* used.
*
* Revision 1.18 1998/01/13 09:17:03 ariels
* Added P_ERROR flag to prm_argv call. This causes all "unused"
* arguments (anything unparsed at the end, i.e. errors!) to be reported.
*
* Revision 1.17 1998/01/08 13:38:28 ariels
* Make parameter block (prm) static in align_cprof.c, rather than an
* externally supplied set of values.
*
* Remove prm from global variable list, and call create_align_prm()
* without a return value.
*
* Revision 1.16 1998/01/04 12:59:42 ariels
* Quit after last contig listed in the "desired" file (when using a
* "desired" file).
*
* Revision 1.15 1998/01/02 13:19:41 avner
* make 'p_ests = n' be default.
*
* Revision 1.14 1997/12/29 22:36:00 avner
* Defining 'bugfix1'. Don't know how come it wasn't defined previously.
*
* Revision 1.13 1997/12/29 16:03:19 eval
* Adding rp_mode to the arg list.
*
* Revision 1.12 1997/12/29 13:04:59 ariels
* fflush(cview_file) at every iteration of the main loop, so we don't
* get the cview file chopped off just because we crashed 5 minutes after
* finishing a contig but didn't write enough of the next.
*
* Revision 1.11 1997/12/29 12:45:28 ariels
* Print sign for repeats (avner)
*
* Revision 1.10 1997/12/22 08:07:12 ariels
```

main.c

```
* Read rich-Fasta files sorted by cluster, then contig, rather than just
* contig. New semantics for "first=", "last=" and "desired=":
* CLUSTER.CONTIG means just that; "CLUSTER" means the entire cluster.
*
* Revision 1.9 1997/12/20 23:04:54 avner
* Adding repeat warning in report line. make 'found_repeat_in_consensus'
* global for that purpose.
*
* Revision 1.8 1997/12/20 21:19:50 avner
* Adding features for the sake of simulations (automatic production of
* pairs-file if not exists, and checking for repeats in the target sequence).
* moving deal_with_singleton to 'io.c'.
*
* Revision 1.7 1997/12/17 15:36:32 ariels
* Support for cluster_view output.
*
* Revision 1.6 1997/12/14 22:37:28 avner
* get rid of variables dealing with the est-penalty accounting.
*
* Revision 1.5 1997/12/14 14:20:35 ariels
* Added 'splicemarks' flag, which adds #SP marks to the rich-Fasta
* transcripts (in transc_db), which give the exact position of each
* proposed splice location. Specifically, at the end of the extent of
* each node in the transcript we place a '#SP pos <out:in>' marker,
* where pos is the position in the transcript, out the out-degree of the
* node we're leaving, and in the in-degree of the node we're entering.
*
* Default (in prologue()) is splicemarks=n, which leaves the transcript
* database in the old format.
*
* Revision 1.4 1997/12/10 22:08:47 avner
* *** empty log message ***
*
* Revision 1.3 1997/12/03 14:38:57 ariels
* fflush(stdout) at the end of the main loop, so we have a reasonably
* up-to-date version of the standard output even when redirecting to a
* file.
*
* Revision 1.2 1997/11/30 08:01:09 ariels
* Added ChangeLog comment and static rcsid string.
*
* */
static char rcsid[]="$Id: main.c,v 1.54 1998/06/30 22:21:14 avner Exp $";

#include <time.h>
#include <stdio.h>
#include "prm.h"
#include "build_graph.h"
#include "repeats.h"
#include "error.h"
#undef FLIPPER
#include "align_prm.h"

/* globals */
FILE *variants_file, *cview_file, *transc_file, *graph_file;
int bound, cutoff, gapext, gapop, match, mismatch, lgapext, lgapop;
int cprof_bound, cprof_lgapop, phase1_id_bound, phase2_id_bound;
int gb_ver, need_support_len, indep_node_len, align_intersec_low;
int align_intersec_high, min_size, max_size, local_olap_mode, max_seq_len;
```

main.c

```

/* flags */
char p_ests, p_seq, p_align, p_est_error, splicemarks, p_error_data,
interactive, use_clone_data, from_variants;
char post_process, p_graph_cpofs, p_profile, check_repeats, p_graph_operations,
order_mechanism, rp_mode, dirty_tails, with_rna, assembly_unit, node_path_mode;
char flip_interactively;
char desired_file_name[UNIX_FNAME_LEN];
char contig_name[CLUSTER_NAME_SIZE], cluster_name[CLUSTER_NAME_SIZE];
char first[CLUSTER_NAME_SIZE], last[CLUSTER_NAME_SIZE];

/* Global memory-use variables for cprof alignment */
extern long cprof_align_max_mem, cprof_align_max_tbl_sz;

char cmp_res[UNIX_FNAME_LEN], out_file[UNIX_FNAME_LEN], keyword_file[100];
char *organism=NULL, *prefix=NULL;
char date[16], dont_assemble[30];
int found_repeat_in_consensus, directed_cycle;
double straight_repeat_max_sim, inverted_repeat_max_sim;

static void simulation_prologue(char *cmp_res, char *infile);
static void mark_dont_assemble(char dont_assemble[]);

/* ===== */
static void prologue(int argc, char *argv[], fasta_file *seqfile,
FILE **desired_file)
{
    long phys_mem = get_phys_mem(), limit_mem = get_max_mem(), mem;

    float change_prob, insert_prob, delete_prob;
    float dirty_change_prob, dirty_insert_prob, dirty_delete_prob;
    float clean_change_prob, clean_insert_prob, clean_delete_prob;

    char name[150], dir[UNIX_FNAME_LEN], fname[UNIX_FNAME_LEN];
    char variants_file_name[UNIX_FNAME_LEN];
    char transc_fname[UNIX_FNAME_LEN];
    char graph_file_name[UNIX_FNAME_LEN];
    char org_key[50] = "human"; /* default */
    char local_mode;

    struct {
        char key[50];
        char organism[100];
        char prefix[5];
    } org_tbl[] = {
        {"human", "Homo sapiens (human)", "CCH"},
        {"mouse", "Mus musculus (mouse)", "CCM"}
    };

    int org_tbl_sz = sizeof(org_tbl)/sizeof(*org_tbl);
    int i;
    char dt_fmt[80];
    struct tm *ttime;
    time_t ctime = time(NULL);
    char months[12][4] = {"JAN", "FEB", "MAR", "APR", "MAY", "JUN",
                        "JUL", "AUG", "SEP", "OCT", "NOV", "DEC"};

    /* Set up memory use for cprof */
    if (phys_mem)
        printf("Physical memory is %ldkb\n", phys_mem/1024);
    else
        printf("Couldn't get physical memory\n");
    if (limit_mem)

```

main.c

```

printf("Per-process memory limit is %ldkb\n", limit_mem/1024);
else
    printf("Couldn't get per-process memory limit\n");
mem = phys_mem < limit_mem ? phys_mem : limit_mem;
if (mem > (8<<20)) {
    /* Set 'reasonable' memory use defaults */
    mem -= 8<<20;
    /* Skip 8MB for various wastage */
    cprof_align_max_mem = mem*0.45;
    cprof_align_max_tbl_sz = mem*0.45;

    if (cpof_align_max_mem > MAX_CPROF_ALIGN_MAX_MEM)
        cprof_align_max_mem = MAX_CPROF_ALIGN_MAX_MEM;
    if (cpof_align_max_tbl_sz > MAX_CPROF_ALIGN_MAX_TBL_SZ)
        cprof_align_max_tbl_sz = MAX_CPROF_ALIGN_MAX_TBL_SZ;
}
printf("Quadratic-space alignment will use %ldkb; "
"log-likelihood speedup table will use %ldkb\n",
cpof_align_max_mem/1024, cprof_align_max_tbl_sz/1024);

/* Set up "local-time" format */
lttime = localtime(&ctime);
sprintf(dt_fmt, "dt = %02d-%03s-%04d %0s", lttime->tm_mon, 1900+lttime->tm_year);
for (i=0; i<argc; i++)
    printf("%s ", argv[i]);
fflush(stdout);
prn_argv(argc, argv, P_PRINT | P_HELP /* broken prn_argv! | P_ERROR */);
! directory",
.dir,
"input file name",
.in,
"frame",
"cmp_res",
! file containing sequences to compare",
cmp_res,
"out",
! output file name (when post process is on)",
out_file,
"transc file",
! name of transcript db",
transc_fname,
"keyword file",
! name of output keyword file",
keyword_file,
"cvview file",
! cluster view filename",
cv_fname,
"graph file",
! graph output filename",
graph_file_name,
graph_file_name,
"desired_file",
! desired cluster's numbers file",
desired_file_name,
"variants_file",
! variant output filename",
variants_file_name,
variants_file_name,

```

main.c

A-121

```

"from variants
! doing just the second phase",
&from_variants,
"interactive
! big intersections solving manually",
&interactive,
"first
first cluster to analyze",
first,
"last
last cluster to analyze",
last,
"min_size
! minimal cluster size to analyze",
&min_size,
"max_size
! maximal cluster size to analyze",
&max_size,
"order
! (a(utomatic)/s(traight)/m(annual)) order",
&order_mechanism,

"bound
! minimal score for alignment",
&bound,
"cutoff
! cutoff plain, defines hills",
&cutoff,
"phase1_id_bound
! alignment bound for variant phase",
&phase1_id_bound,
"phase2_id_bound
! alignment bound for main phase",
&phase2_id_bound,

"match
! score for match",
&match,
"mismatch
! penalty for mismatch",
&mismatch,
"gapop
! additive constant for gaps",
&gapop,
"gapext
! multiplicative constant for gaps",
&gapext,
"lgapext
! multiplicative constant for long gaps",
&lgapext,
"lgapop
! additive constant for long gaps",
&lgapop,
"cpof_bound
! bound in cprof_alignments",
&cpof_bound,
"cpof_lgapop
! additive constant for long gaps in cprof2cpof",
&cpof_lgapop,
"change_prob
= .02

```

main.c

```

! probability of base change",
&change_prob,
"insert_prob
! probability of base insertion",
&insert_prob,
"delete_prob
! probability of base deletion",
&delete_prob,
"rna_change_prob
! probability of RNA base change",
&clean_change_prob,
"rna_insert_prob
! probability of RNA base insertion",
&clean_insert_prob,
"rna_delete_prob
! probability of RNA base deletion",
&clean_delete_prob,
"dirty_change_prob
! probability of dirty base change",
&dirty_change_prob,
"dirty_insert_prob
! probability of dirty base insertion",
&dirty_insert_prob,
"dirty_delete_prob
! probability of dirty base deletion",
&dirty_delete_prob,
"local_mode
! use local alignments (not overlap)",
&local_mode,
"need_support_len
! length for nodes to be supported",
&need_support_len,
"indep_node_len
! up to this len need support",
&indep_node_len,
"align_intersec_low
! (low end of) the longest region that can be multiple aligned",
&align_intersec_low,
"align_intersec_high
! (high end of) the longest region that can be multiple aligned",
&align_intersec_high,
"gb_ver
! genbank version number",
&gb_ver,
"p_error_data
! data about errors results",
&p_error_data,
"p_est
! print ests in cluster",
&p_est,
"p_graph_operations
! no/yes/variant-graph-too",
&p_graph_operations,
"p_seq
!",
&p_seq,
"p_alignstr
!",
&p_alignstr,
"p_est_error
= y

```

main.c

A-122


```

! ",
&p_est_error,
! "p_profile
= n

! p_graph_cprofs
= n
! y/n prints the cprof of each node of the graph",
&p_graph_cprofs,
! "post_process
= y
! No/Yes/interesting-only",
&post_process,
! "splicemarks
= y
! No/Yes (annotates transc_file)",
&splicemarks,
! "check_repeats
= v
! check repeats in: consensus/variants/multiple hits",
&check_repeats,
! "use_clone_data
= y
! incorporate clone pairing",
&use_clone_data,
! "organism
= human
! human or mouse (for output)",
org_key,
! "rp_mode
= s
! repeat mode(cyclic graph): all/build/transcript/secondary",
&rp_mode,
! "node_path_mode
= n
! node_path n/y",
&node_path_mode,
! "dont_assemble
=
! rna/biggest_rna<prefix> (only show certain sequences",
dont_assemble,
! "dirty_tails
= y
! use dirty tails of sequences",
&dirty_tails,
! "with_rna
= y
! process: n - no RNA, y - est+RNA, o - only RNA",
&with_rna,
! "max_seq_len
= 20000
! maximal sequence length to process (ignore if -1)",
&max_seq_len,
! "assembly_unit
= n
! n for contig, u for cluster a for the whole rf file",
&assembly_unit,
! "flip_interactively
= n
! y - ask whether to flip each sequence",
&flip_interactively,
! "dt_fmt, date,
EOLIST);

local_olap_mode=(local_mode=='y')?LOCAL_MODE:0;
init_matrix (match, mismatch, gapext, gapop);
init_long_gaps (lgapext, lgapop);

cview_file = NULL;
transc_file = NULL;
variants_file = NULL;
*desired_file = NULL;

```

main.c

```

create_align_prm(change_prob, delete_prob, insert_prob,
clean_change_prob, clean_delete_prob, clean_insert_prob,
dirty_change_prob, dirty_delete_prob, dirty_insert_prob,
cprof_lgapop);

sprintf(name, "%s/%s", dir, fname);
if ((*seqfile=rich_fasta_open(name))==NULL)
err("can't open sequence file %s", name);
if (strcmp(desired_file_name, "none")) {
if ((*desired_file=fopen(desired_file_name, "r"))==NULL)
printf("can't open desired file %s\n", desired_file_name);
}
if (strcmp(variants_file_name, "none")) {
if ((*variants_file=fopen(variants_file_name, "w"))==NULL)
printf("can't open variants file %s\n", variants_file_name);
}
if (strcmp(cv_fname, "none")) {
if ((*cview_file = fopen(cv_fname, "a")) == NULL)
printf("can't open cluster_view files %s\n", cv_fname);
}
if (strcmp(transc_fname, "none")) {
if ((*transc_file = fopen(transc_fname, "a")) == NULL)
printf("can't open transc_file files %s\n", transc_fname);
}
if (strcmp(graph_file_name, "none")) {
if ((*graph_file = fopen(graph_file_name, "a")) == NULL)
printf("can't open graph_file %s\n", graph_file_name);
}
for(i=0; i<org_tbl_sz; i++)
if (strcmp(org_key, org_tbl[i].key) == 0) {
organism = strdup(org_tbl[i].organism);
prefix = strdup(org_tbl[i].prefix);
break;
}
if (i == org_tbl_sz)
err("Unknown organism \"%s\"", org_key);
if (strcmp(cmp_res, "none") != 0)
simulation_prologue(cmp_res, name);

static void simulation_prologue(char *cmp_res, char *infile)
{
fasta_file ff;
rich_fasta_seq_ptr rfp;

if ((ff = rich_fasta_open(cmp_res)) == NULL ||
(rfp = rich_fasta_read_seq(ff, ASSEMBLY_MODE)) == NULL)
err("Can't get target sequence. ignoring...\n");
check_repeat(rfp->original_seq, "target sequences");
}

int get_next_desired(FILE *f, char *desired)
{
if (fscanf(f, "%s", desired)>0)
return 1;
else return 0;
}
/*

```

main.c

```

* Compare two contig identifiers, in CLUSTER[.CONTIG] format. If
* CONTIG is empty, CLUSTER will match *any* contig identifier with
* the same cluster name.
*
* Return <0, 0, >0 to indicate first argument comes before, together
* with or after the second argument.
*/

```

```

int cid_cmp(char *c1, char *c2)
{
    char *d1 = strdup(c1), *d2 = strdup(c2);
    char *e1, *e2;
    int res;

    if (e1 = strchr(d1, '.'))
        *e1++ = '\0';
    if (e2 = strchr(d2, '.'))
        *e2++ = '\0';

    if (! (res = strcmp(d1, d2))) && e1 && e2)
        res = strcmp(e1, e2);
    free(d1);
    free(d2);
    return res;
}

/*
* Mark a certain sequence that we do not wish to participate in the assembly.
* It will, however, be aligned with the consensus. Now, it is the longest
* rna. If found a sequence with the property returns 1, 0 otherwise
*/
static void mark_dont_assemble(char dont_assemble[])
{
    int i;
    if (strlen(dont_assemble) == 0) return;
    if (strcmp(dont_assemble, "rna")) {
        for(i=0; i<est_table__size(); i++)
            if (est_table__is_rna(i))
                est_table__set_test_case(i);
    }
    else
        if (strcmp(dont_assemble, "biggest_rna")) {
            int longest_index=-1;
            for(i=0; i<est_table__size(); i++){
                if (est_table__is_rna(i) && longest_index == -1)
                    longest_index = i;
            }
            if ((longest_index != -1) &&
                est_table__len(i) > est_table__len(longest_index))
                longest_index = i;
        }
        if (longest_index != -1)
            est_table__set_test_case(longest_index);
    }
    else { /* mark sequences whose name start with the string <dont_assemble> */
        for(i=0; i<est_table__size(); i++)
            if (strcmp(id(i), dont_assemble, strlen(dont_assemble)) == 0)
                est_table__set_test_case(i);
    }
}

void clean_workspace(splice_graph **graph)

```

main.c

```

{
    directed_cycle = 0;
    est_table__clean();
    if (*graph)
        splice_graph_free(*graph);
    *graph=NULL;
}

void print_title(void)
{
    printf("#####\n");
    if(assembly_unit!='u')
        printf("### begin %10s %10s (size = %5d) ###\n",
            cluster_name, contig_name, est_table__size());
    else
        printf("### begin cluster %10s (size = %5d) ###\n",
            cluster_name, est_table__size());
    printf("#####\n");
}

void print_online_report(splice_graph *graph)
{
    char straight_repeat_str[20], inverted_repeat_str[20];
    int undirected_cycle, disconnected;
    if (graph) {
        printf(straight_repeat_str, "repeats-bin(%3.0f)", straight_repeat_max_sim);
        printf(inverted_repeat_str, "flipper-bin(%3.0f)", inverted_repeat_max_sim);
        undirected_cycle = splice_graph__is_undirected_cycles(graph);
    }
    if(assembly_unit!='u')
        printf("report contig %s %s (%3d seqs) : ".cluster_name, contig_name,
            est_table__size());
    else
        printf("report cluster %s (%3d seqs) : ".cluster_name, est_table__size());
    if (graph && !erred()) {
        printf
            ("good, %2d nodes %d->%c->%d %10s %s %s", graph->size,
             splice_graph__num_initial(graph),
             directed_cycle?'c':'',
             splice_graph__num_terminal(graph),
             directed_cycle?'to-repeats-bin(c)':'',
             undirected_cycle?
             (splice_graph__num_initial(graph) <= 1 &&
              splice_graph__num_terminal(graph) <= 1)?
              "(very-interesting)": "(interesting)": "",
             found_repeat_in_consensus & STRAIGHT_REPEAT ?
             straight_repeat_str : "",
             found_repeat_in_consensus & INVERTED_REPEAT ?
             inverted_repeat_str : "");
    }
    else {
        printf("bad: ");
        report_err_online(stdout, " ");
        putchar('\n');
    }
    printf("\n\n");
}

/*=====
main(int argc, char *argv[])
{

```

main.c

```

FILE *desired_file=NULL;
fasta_file seqfile;
splice_graph *graph = NULL;
float est_ins,est_del,est_mis,est_not,err_cor;
char desired[2*CLUSTER_NAME_SIZE];
char cid[2*CLUSTER_NAME_SIZE];
int i;

/* Set buffering of standard output to LINE buffering */
if (setvbuf(stdout, NULL, _IO_LBF, BUFSIZ))
    err("failed to line-buffer STDOUT");

prologue(argc,argv,&seqfile,&desired_file);

strcpy(contig_name,"STAM");
cid[0] = '\0';
if (desired_file)
    get_next_desired(desired_file,desired);
else
    desired[0]=0;
if (cview_file)
    cv_write_header(cview_file);
while (contig_name[0]!='\0') {
    fflush(stdout);
    if (cview_file)
        fflush(cview_file);
    if (transc_file)
        fflush(transc_file);

    /* Reset data */
    clean_workspace(&graph);

    /* Read a new cluster (contig) */
    read_sequences(seqfile,contig_name,cluster_name,assembly_unit);
    if (gb_ver >= 104 && cluster_name[0])
        if (strcmp(contig_name, "UNKNOWN") != 0 || assembly_unit != 'u')
            sprintf(cid, "%s.%s", cluster_name, contig_name);
        else {
            /* Fake contig name from cluster */
            strcpy(cid, cluster_name);
        }
    else
        strcpy(cid, contig_name);

    /* Skip contig with RNA/est if needed */
    if (with_rna == 'n') {
        /* skip RNA */
        for (i=0; i<est_table__size(); i++)
            if (est_table__is_rna(i))
                break;
        if (i<est_table__size()) continue;
    }
    if (with_rna == 'o') {
        /* skip contig with no RNA */
        for (i=0; i<est_table__size(); i++)
            if (est_table__is_rna(i))
                break;
        if (i==est_table__size()) continue;
    }

    /* Skip contigs with too long sequence */

```

main.c

```

if (max_seq_len != -1) {
    for (i=0; i<est_table__size(); i++)
        if (est_table__len(i) >= max_seq_len)
            break;
    if (i<est_table__size()) continue;
}

/* Skip clusters which don't meet our demands */
if (est_table__size()<min_size) continue;
if (est_table__size()>max_size) continue;

/* Is this contig on a specified "desired" list? */
if (desired[0]) {
    while (cid_cmp(desired,cid)<0 && get_next_desired(desired_file,desired))
        if (feof(desired_file))
            break;
    if (cid_cmp(desired,cid) != 0)
        continue; /* This contig not desired */
}

/* End loop if we're past the last cluster to process */
if (strlen(last) && cid_cmp(cid,last)>0)
    break;

/* Skip everything before the first cluster */
if (strlen(first) && cid_cmp(first, cid) > 0)
    continue;

/* Deal with degenerate contigs */
if (est_table__size()==1) {
    deal_with_singletons(cview_file,contig_name);
    graph=NULL;
    continue;
}

if (flip_interactively == 'y') {
    int i;
    char ans[256];
    for (i=0; i<est_table__size(); i++) {
        printf("Flip sequence %s? (y/n)\n", id(i));
        gets(ans);
        if (tolower((unsigned char)ans[0]) == 'y')
            est_table__invert_seq(i);
    }

    /* Assemble contig */
    first[0]='\0';
    mark_dont_assemble(dont_assemble);
    print_title();
    if (use_clone_data == 'y')
        est_table__pair_clones();
    graph = build_graph();
    if (graph && post_process != 'n')
        cluster_post_process(graph, contig_name, cluster_name);
    if (warned())
        report_warn(stdout, "Warning: ");
}

```

main.c

A-125

Sun Aug 9 10:33:22 1998

Listing for Adam Santiel

```
print_contig_online_report(graph);
clear_err();
clear_warn();
}
rich_fasta_close(seqfile);
}
```

main.c

```

/* * Routines to access amount of available and physical memory
* * VERY platform-dependent, alas...
* *
* *
#include <stdio.h>

/* * Get amount of physical memory */
#ifdef __sgi

/* * SGI -- read it from the hardware inventory */
#include <invent.h>

static int phys_mem_scanner(inventory_t *inv, void *dummy)
{
    return (inv->inv_class == INV_MEMORY && inv->inv_type == INV_MAIN_MB) ?
        inv->inv_state :
        0;
}

long get_phys_mem(void)
{
    int res = scaninvent(phys_mem_scanner, NULL);

    if (res < 0) {
        fprintf(stderr, "ERROR: Hardware inventory scan failed\n");
        exit(-1);
    }
    return res * (1<<20);
}

#elif defined(__sun)

/* * Sun -- use sysconf(3C) */
#include <unistd.h>

long get_phys_mem(void)
{
    int n_pages = sysconf(_SC_PHYS_PAGES);
    int page_sz = sysconf(_SC_PAGE_SIZE);

    return (long) n_pages * page_sz;
}

#elif defined(__alpha)

/* * DREC -- use getsysinfo(3), of course... */
#include <sys/sysinfo.h>

long get_phys_mem(void)
{
    int n_kbs;

    if (getsysinfo(GSI_PHYSMEM, (char*)&n_kbs, sizeof(n_kbs)) == -1) {
        perror("get_phys_mem");
        exit(-1);
    }
    /* REALLY impossible */
}

```

mem.c

```

    return n_kbs*1024;
}

#ifdef defined(__linux)

/* * Linux -- use sysinfo(2), of course */
#include <linux/kernel.h>
#include <linux/sys.h>

long get_phys_mem(void)
{
    struct sysinfo mysys;

    if (sysinfo(&mysys) == -1) {
        perror("get_phys_mem");
        exit(-1);
    }
    return mysys.totalram;
}

#else
/* *
/* * New type of system. As the examples above show, it is perfectly
/* * simple to write a portable RAM-counter. The only problem is that you
/* * always need to call a different function with different arguments.
/* *
/* * Good Luck!
/* * (you'll need it)
/* *
long get_phys_mem(void)
{
    return 0;
}
#endif

/* * Return (soft) total RAM resource limit */
#include <sys/resource.h>
long get_max_mem(void)
{
    struct rlimit ram_limit;

    /* * Asking for RLIMIT_AS doesn't seem to work on DRECs, alas */
    if (getrlimit(RLIMIT_DATA, &ram_limit) == -1) {
        perror("get_max_mem");
        exit(-1);
    }
    return ram_limit.rlim_cur;
}

#ifdef TEST
main()
{
    printf("Detected %ld bytes; limit %ld bytes\n", get_phys_mem(), get_max_mem());
    ;
    exit(-1);
}
#endif

```

mem.c

A-127


```

/* $Log: olap_graph.c,v $
 * Revision 1.4 1998/02/09 06:50:40 avner
 * (For flipper) Only one function of the old graph still relevant. The
 * only thing it does is allowing interactive choice for the order. Remember
 * that the automatically generated order of the past is the straight order
 * of the present, since flipper order the input file.
 *
 * Revision 1.3 1998/02/08 12:18:47 avner
 * Clean the dust that was in the code, in preparation to it's abuse by flipper.
 * Also add
 * comments.
 *
 * Revision 1.2 1997/11/30 08:01:44 ariels
 * Added ChangeLog comment and static rcsid string.
 */

static char rcsid[] = "$Id: olap_graph.c,v 1.4 1998/02/09 06:50:40 avner Exp $";

#include "olap_graph.h"
#include "est_table.h"

void order_ests(int *order_arr);

extern char order_mechanism;

/*=====*/
/*
 * determine order in which fragments will be processed.
 * Normally the order is produced by iteratively taking a fragment
 * that align the most (to some degree of approximation) with the component
 * of the previous fragments/
 */

void order_ests(int *order_arr)
{
    int next_in_order, ordinal, j;

    if (order_mechanism != 'm') {
        for (ordinal=0; ordinal<est_table__size(); ordinal++)
            order_arr[ordinal]=ordinal;
    }

    if (order_mechanism=='m')
    {
        printf("please insert the desired order for the ests ");
        for (j=0; j<est_table__size(); j++)
            scanf("%d", &order_arr[j]);
    }
}

```



```

/* $Log: output.c,v $
 * Revision 1.2 1997/11/30 08:02:12 ariels
 * Added ChangeLog comment and static rcsid string.
 */

static char rcsid[] = "$Id: output.c,v 1.2 1997/11/30 08:02:12 ariels Rel $";

#include "olap_graph.h"
#include "est_table.h"

/*extern edge_in_olap_graph graph_table[MAX_CLUSTER_SIZE][MAX_CLUSTER_SIZE];*/
print_predec_graph()
{
    int i,j;
    for(i=0; i< est_table__size(); i++)
        for (j=0; j<est_table__size(); j++)
        {
            if (is_contained(i)||is_contained(j) || !graph_table[i][j].edge) continue;
            if (graph_table[i][j].predecessor) printf("\n%d(%s)->%d(%s)",j,id(j),i,
            id(i));
            if (graph_table[i][j].left_contain) printf("\n%d(%s){%d(%s)}",i,id(i),j,
            id(j));
        }
    int is_contained (int idx)
    {
        int i;
        for (i=0; !graph_table[i][idx].left_contain && i<est_table__size(); i++)
            return (i<est_table__size());
    }
}

print_edge(edge_in_olap_graph edge, int first, int second)
{
    int tmp;
    printf("\n%d.%d%c",first,second,edge.left_in_algn?'+':'-');
    edge.right_in_algn?'+':'-';
    if (est_table__is_inverted(first))
    {
        tmp=edge.st1;
        edge.st1=len(first)-1- edge.end1;
        edge.end1=len(first)-1- tmp;
        tmp=edge.st2;
        edge.st2=edge.end2;
        edge.end2=tmp;
    }
    if (est_table__is_inverted(second))
    {
        edge.st2=len(second)-1- edge.st2;
        edge.end2=len(second)-1- edge.end2;
    }
    printf("    %d.%d len(%d) | %d.%d len(%d)",

```

output.c

```

        edge.st1.edge.end1, len(first), edge.st2, edge.end2, len(second));
    }

    /*
    print_inverses()
    {
        int i,j;

        printf("the inverted alignments are \n");
        for(i=0; i< est_table__size()-1; i++)
            for (j=i+1; j<est_table__size(); j++)
                if (est_table__neighbors(i,j) && est_table__neighbors(i,j)->inverted)
                    printf("%d <-> %d\n",i,j);
    }

    void print_colors()
    {
        int i;

        for (i=0; i<est_table__size(); i++)
            if (est_table__get_inverse_color(i)==2)
                printf("%d(%s) is reversed\n",i,id(i));
    }

    void print_ests_order(int *order_arr)
    {
        int i;

        for (i=0; i<est_table__size(); i++)
            printf("%d ",order_arr[i]);
        printf("\n");
    }
}

```

output.c

A-129


```

        problematic_inversion++;
    }
    printf ("inverting in score %d, %d problematic (size %d)\n", inversion_score,
        problematic_inversion, est_table__size());
}

/*=====
void get_pairs_information (FILE *pairs_file, char *mc)
{
    extern int simulation_mode;
    int loc1, loc2;
    char *inv;
    hilltop *list, *ht;

    if (p_pairs == 'y')
        printf ("Doing pairwise alignments on %d sequences\n",
            est_table__size());
    for (loc1 = 0; loc1 < est_table__size()-1; loc1++)
        for (loc2 = loc1+1; loc2 < est_table__size(); loc2++) {
            /* straight */
            list = HillTops (est_table__original_seq(loc1),
                est_table__original_seq(loc2),
                0, cutoff, bound);
            if (ht && est_table__contig(loc1) != est_table__contig(loc2))
                err("alignment detected between %s in %s and %s in %s",
                    id(loc1), est_table__seq_data(loc1)->cluster,
                    id(loc2), est_table__seq_data(loc2)->cluster);
            for (ht = list; ht != NULL; ht = ht->next) {
                if (p_pairs == 'y')
                    printf ("%s %s %s + %d %d %d\n",
                        mc, est_table__seq_data(loc1)->cluster, id(loc1), id(loc2),
                        ht->x0, ht->xt, ht->y0, ht->yt);
                if (strcmp("wyca", write_pairs_file))
                    fprintf (pairs_file,
                        "%s %s %s + %d %d %d\n",
                        mc, est_table__seq_data(loc1)->cluster, id(loc1), id(loc2),
                        ht->x0, ht->xt, ht->y0, ht->yt);
                est_table__add_edge (loc1, loc2, ht->x0, ht->xt, ht->y0, ht->yt);
            }
            destroy_hilltop_list (&list);
        }
    /* inverted */
    inv = subseq (est_table__original_seq(loc2),
        est_table__len(loc2)-1, 0);
    list = HillTops (est_table__original_seq(loc1), inv, 0, cutoff, bound);
    if (ht && est_table__contig(loc1) != est_table__contig(loc2))
        err("inverse alignment detected between %s in %s and %s in %s",
            id(loc1), est_table__seq_data(loc1)->cluster,
            id(loc2), est_table__seq_data(loc2)->cluster);
    for (ht = list; ht != NULL; ht = ht->next) {
        if (p_pairs == 'y')
            printf ("%s %s %s - %d %d %d\n",
                mc, est_table__seq_data(loc1)->cluster, id(loc1), id(loc2),
                ht->x0, ht->xt, ht->y0, ht->yt);
        if (strcmp("wyca", write_pairs_file))
            fprintf (pairs_file,
                "%s %s %s - %d %d %d\n",

```

parse.c

```

        mc, est_table__seq_data(loc1)->cluster, id(loc1), id(loc2),
        ht->x0, ht->xt, ht->y0, ht->yt);
    est_table__add_edge (loc1, loc2, ht->x0,
        est_table__len(loc2)-1, ht->y0,
        ht->xt, est_table__len(loc2)-1, ht->yt);
    }
    destroy_hilltop_list (&list);
    free (inv);
}

static char last_line[LINE_SIZE];

void parse_pairs_information (FILE *pairs_file, char *mc)
{
    int loc1, loc2, x0, xt, y0, yt, read_ok;
    char c, line[LINE_SIZE], estname2[MAX_KEY_LEN];
    char cluster_read[CLUSTER_NAME_SIZE], meta_cluster_read[CLUSTER_NAME_SIZE];
    char meta_cluster[CLUSTER_NAME_SIZE];

    extern char bugfix1;

    printf ("reading pairs info of %d sequences\n", est_table__size());
    do {
        if (last_line[0] == 0)
            if (fgetc(line, LINE_SIZE, pairs_file) == NULL)
                break;
        if (last_line[0] != 0) {
            strcpy(line, last_line);
            last_line[0] = 0;
        }
        if (gb_ver >= 102)
            if (gb_ver >= 104)
                read_ok = sscanf(line, "%s %s %s %c %d %d %d %d",
                    meta_cluster_read, cluster_read,
                    estname1, estname2, &c, &x0, &xt, &y0, &yt) == 9;
            else
                read_ok = sscanf(line, "%d %s %s %c %d %d %d %d",
                    cluster_read, estname1, estname2,
                    &c, &x0, &xt, &y0, &yt) == 8;
        else
            read_ok = sscanf(line, "%s %s %s %c %d %d %d %d", cluster_read,
                estname1, estname2, &c, &x0, &xt, &y0, &yt) == 8;
        if (! read_ok)
            err("parse_pairs_information: internal error. cannot parse line '%s'",
                line);
        /*
        if (gb_ver >= 104)
            sprintf(cid_read, "%s.%s", meta_cluster_read, cluster_read);
        else
            strcpy(cid_read, cluster_read);
        */
        if (gb_ver >= 104)
            strcpy(meta_cluster, meta_cluster_read);
        else
            strcpy(meta_cluster, cluster_read);
    } while (1);
}

```

parse.c

Listing for Adam Sartiel Sun Aug 9 10:33:23 1998

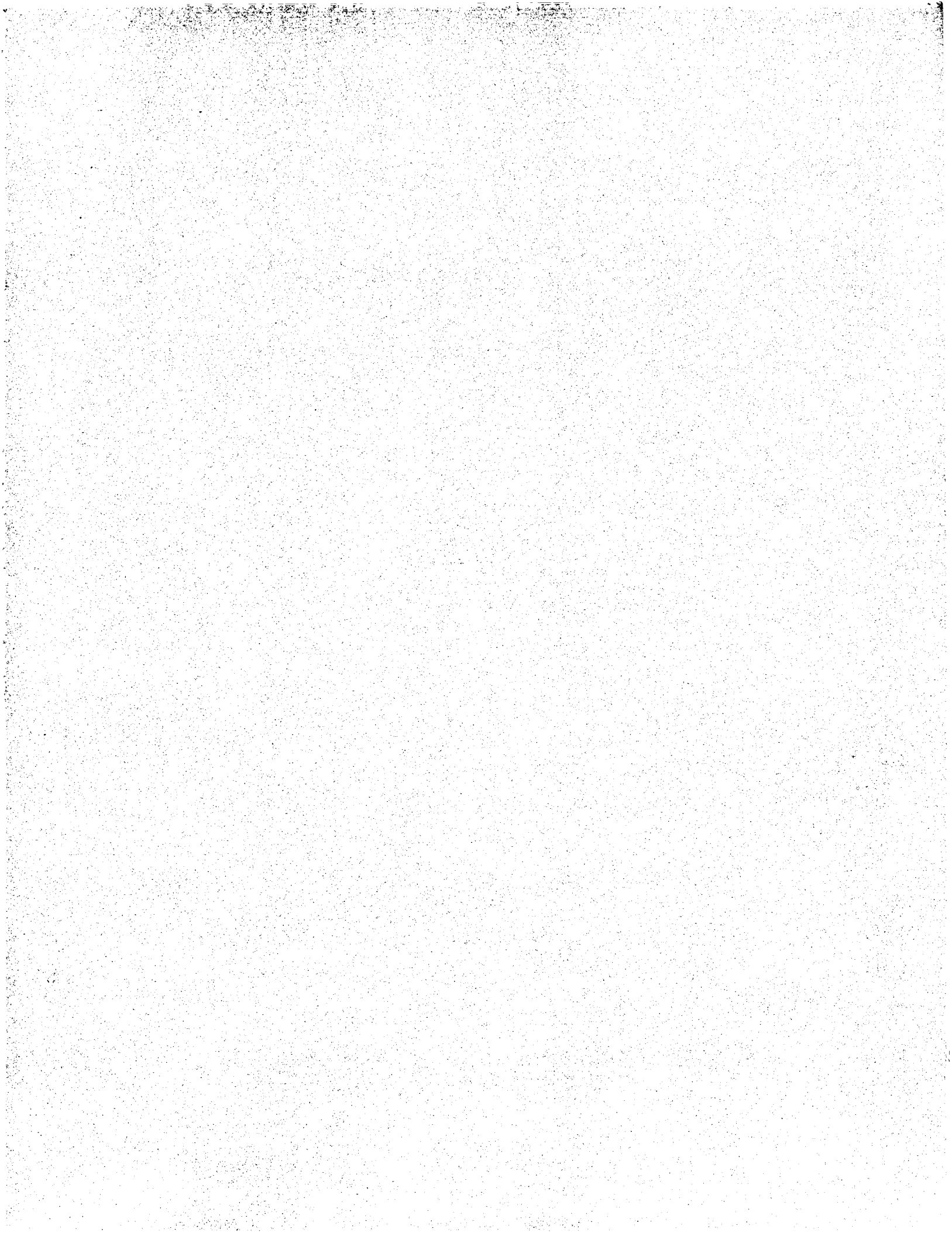
```

#ifdef DESCENDING_ORDER
    if (strcmp(meta_cluster, mc) > 0)
#else
    if (strcmp(meta_cluster, mc) < 0)
#endif
    continue;

#ifdef DESCENDING_ORDER
    if (strcmp(meta_cluster, mc) < 0)
#else
    if (strcmp(meta_cluster, mc) > 0)
#endif
    strcpy(last_line, line);
    else {
        if ((loc1 = look_up(estname1)) == -1)
            err("%s in htops file is not known to be an est", estname1);
        if ((loc2 = look_up(estname2)) == -1)
            err("%s in htops file is not known to be an est", estname2);
        if (p_pairs == 'Y')
            if (gb_ver >= 104)
                printf("%s %s %s %c %d %d %d %d\n",
                    meta_cluster_read, cluster_read,
                    estname1, estname2, c, x0, xt, y0, yt);
            else
                printf("%s %s %s %c %d %d %d\n", cluster_read,
                    estname1, estname2, c, x0, xt, y0, yt);
        if (c == '+') {
            est_table__add_edge(loc1, loc2, x0, y0, xt, yt);
            est_table__add_edge(loc2, loc1, y0, x0, yt, xt);
        }
        else {
            est_table__add_edge(loc1, loc2, x0, est_table__len(loc2)-1-y0,
                xt, est_table__len(loc2)-1-yt);
            est_table__add_edge(loc2, loc1, est_table__len(loc2)-1-yt, xt,
                est_table__len(loc2)-1-y0, x0);
        }
    }
}

#ifdef DESCENDING_ORDER
    while (strcmp(meta_cluster, mc) >= 0);
#else
    while (strcmp(meta_cluster, mc) <= 0);
#endif
}

```




```

/* $Log: profile.c,v $
 * Revision 1.8 1998/02/22 22:03:18 avner
 * Get rid of 'throw_gaps' parameter, who was _always_ 'y', and became
 * obsolete.
 *
 * Revision 1.7 1998/02/22 17:14:11 ariels
 * Change various "harmless" printf()'s to fatal err()'s
 *
 * Revision 1.6 1998/02/17 16:20:29 ariels
 * Fix error message to use err(), not print() (it's an internal error,
 * therefore it must be fatal).
 *
 * Revision 1.5 1998/01/21 07:54:10 ariels
 * Put err() in scope of prototype (add #include "error.h")
 *
 * Revision 1.4 1998/01/20 09:20:28 ariels
 * Fix append_node warning messages (used to mention prepend_node)
 *
 * Revision 1.3 1998/01/08 13:39:09 ariels
 * Use **correct*** routines for varargs error reporting (previously worked
 * fortuitously on alphas because of compiler quirks).
 *
 * Revision 1.2 1997/11/30 08:04:01 ariels
 * Added ChangeLog comment and static rcsid string.
 */

static char rcsid[] = "$Id: profile.c,v 1.8 1998/02/22 22:03:18 avner Exp $";

#include <math.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include "profile.h"
#include "error.h"

extern char p_error_data;
static err_place;
static void seq_prof_node___copy(seq_prof_node *src, seq_prof_node **dup);
seq_prof_node *seq_prof_node_new()
{
    seq_prof_node *seqpn;

    seqpn = (seq_prof_node *) malloc(sizeof(seq_prof_node));
    if (seqpn == NULL)
        err("seq_prof_node_new: memory failure trying to allocate for 'seqpn'");
    seqpn->a = seqpn->c = seqpn->g = seqpn->t = seqpn->gap = seqpn->sum = 0.0;
    seqpn->next = NULL;
    return seqpn;
}

/*****
seq_prof_node *seq_prof_node_create (char ch)
{
    seq_prof_node *seqpn;

```

```

case 'N':
case 'X':
    seqpn->g = 0.25;
    seqpn->c = 0.25;
    seqpn->t = 0.25;
    seqpn->a = 0.25;
    break;
default:
    err("@Warning: unrecognized character %c in seq_prof_node_create.\n",ch);
}
seqpn->sum = seqpn->a+seqpn->c+seqpn->g+seqpn->t;
return seqpn;
}

/*****
void seq_prof_node_free (seq_prof_node *seqpn)
{
    if (seqpn)
        free(seqpn);
}

/*****/
void seq_prof_node_update (seq_prof_node *seqpn, char ch)
{
    switch (ch) {
case 'A':
    seqpn->a += 1.0;
    break;
case 'C':
    seqpn->c += 1.0;
    break;
case 'G':
    seqpn->g += 1.0;
    break;
case 'T':
    seqpn->t += 1.0;
    break;
case 'W':
    seqpn->a += 0.5;
    seqpn->c += 0.5;
    break;
case 'K':
    seqpn->t += 0.5;
    seqpn->g += 0.5;
    break;
case 'R':
    seqpn->a += 0.5;
    seqpn->g += 0.5;
    break;
case 'S':
    seqpn->c += 0.5;
    seqpn->g += 0.5;
    break;
case 'W':
    seqpn->a += 0.5;
    seqpn->t += 0.5;
    break;
case 'Y':
    seqpn->t += 0.5;

```

profile.c

```

seqpn->c += 0.5;
break;
case 'B':
    seqpn->t += 0.333;
    seqpn->g += 0.333;
    seqpn->c += 0.333;
    break;
case 'D':
    seqpn->t += 0.333;
    seqpn->g += 0.333;
    seqpn->a += 0.333;
    break;
case 'H':
    seqpn->t += 0.333;
    seqpn->c += 0.333;
    seqpn->a += 0.333;
    break;
case 'V':
    seqpn->g += 0.333;
    seqpn->c += 0.333;
    seqpn->a += 0.333;
    break;
case 'N':
case 'X':
    seqpn->g += 0.25;
    seqpn->c += 0.25;
    seqpn->t += 0.25;
    seqpn->a += 0.25;
    break;
case '-':
    seqpn->gap++;
    break;
default:
    err("@Warning: unrecognized character %c in seq_prof_update.\n",ch);
}
seqpn->sum = seqpn->a+seqpn->c+seqpn->g+seqpn->t;
}
/*****/
char seq_prof_node_char (seq_prof_node *seqpn)
{
    /* returns the character represented by the profile node, using simple majority
    Y. in case of gap majority it returns the majority of the a,c,g,t fields but i
    n lower casee.
    */
    #undef F
    #define F(x) (1-1./sqrt(x+1))
    char amb[16] = "XACGRSVTWYHKDBN";
    extern float iter_gap_major;
    float a_p, c_p, g_p, t_p, max_p;

    float sum = seqpn->sum;
    char ch, max_ch = 0;
    float t;
    if (seqpn->gap / (seqpn->gap+sum) > iter_gap_major) return 0;

```

profile.c

```

max_ch = 1;
max_p = a_p = seqpn->a/sum;
if ((c_p = seqpn->c/sum) > max_p) {
    max_ch = 2;
    max_p = c_p;
}
if ((g_p = seqpn->g/sum) > max_p) {
    max_ch = 4;
    max_p = g_p;
}
if ((t_p = seqpn->t/sum) > max_p) {
    max_ch = 8;
    max_p = t_p;
}
ch = max_ch;
t = max_p * F(sum);

if (a_p > t)
    ch |= 1;
if (c_p > t)
    ch |= 2;
if (g_p > t)
    ch |= 4;
if (t_p > t)
    ch |= 8;

if (seqpn->gap / sum > 1.2) return 0;
return amb[ch];
}

char final_seq_prof_node_char (seq_prof_node *seqpn, int *iserror,
                              int *iscorrected) {
    /* returns the character represented by the profile node, using simple majority
    y. in case of gap majority it returns the majority of the a,c,g,t fields but i
    n lower case.
    */
    #undef F
    #define F(x) (1-0.5/sqrt(x+1))

    char amb[16] = "XACMGSRVWYHKDEN";
    extern float final_gap_major;
    float a_p, c_p, g_p, t_p, max_p, max_p_with_gaps;

    float sum = seqpn->sum;
    char ch, max_ch = 0;
    float t;
    *iserror = 0;
    *iscorrected = 0;

    if (seqpn->gap / (seqpn->gap+sum) > final_gap_major)
    {
        *iserror = 1;
        *iscorrected = 1;
    }

```

```

    return ' ';
}
max_ch = 1;
max_p = a_p = seqpn->a/sum;
if ((c_p = seqpn->c/sum) > max_p) {
    max_ch = 2;
    max_p = c_p;
}
if ((g_p = seqpn->g/sum) > max_p) {
    max_ch = 4;
    max_p = g_p;
}
if ((t_p = seqpn->t/sum) > max_p) {
    max_ch = 8;
    max_p = t_p;
}
ch = max_ch;
max_p_with_gaps = max_p * sum / (sum + seqpn->gap);
if (max_p_with_gaps < 0.999) {
    *iserror = 1;
    if (p_error_data == 'y') printf(" profile error at place %d ", err_place);
    if (max_p_with_gaps > 0.65)
    {
        if (p_error_data == 'y') printf("corrected: ");
        *iscorrected = 1;
    }
    else if (p_error_data == 'y') printf("cant correct: ");
    if (p_error_data == 'y') print_seq_prof_node(seqpn);
}
t = max_p * F(sum);

if (a_p > t)
    ch |= 1;
if (c_p > t)
    ch |= 2;
if (g_p > t)
    ch |= 4;
if (t_p > t)
    ch |= 8;
return amb[ch];
}

/*****
seq_prof *seq_prof_new ()
{
    seq_prof * sp;

    sp = (seq_prof *) malloc(sizeof(seq_prof));
    if (sp == NULL)
        err("seq_prof_new: memory failure trying to allocate for sp");
    sp->len = 0;
    sp->first = sp->last = 0;
    return sp;
}

```

```

void profile__copy(seq_prof *src, seq_prof **dup)
{
    seq_prof_node *src_seq_node, **p;
    *dup = seq_prof_new();
    (*dup)->len = src->len;
    p = &(*dup)->first;
    for (src_seq_node = src->first; src_seq_node;
         src_seq_node = src_seq_node->next)
    {
        seq_prof_node__copy(src_seq_node, p);
        if (!src_seq_node->next)
            (*dup)->last = *p;
        p = &((*p)->next);
    }
}

static void seq_prof_node__copy(seq_prof_node *src, seq_prof_node **dup)
{
    *dup = seq_prof_node_new();
    (*dup)->a = src->a;
    (*dup)->c = src->c;
    (*dup)->g = src->g;
    (*dup)->t = src->t;
    (*dup)->sum = src->sum;
    (*dup)->gap = src->gap;
}

/*****
seq_prof *seq_prof_create (char *seq)
{
    /* seq is assumed to be uppercase. bad characters are ignored */
    seq_prof *sp;
    seq_prof_node *seqpn;
    int len = 0;
    char ch;

    sp = seq_prof_new();
    if (!sp)
        return NULL;

    while (ch = *seq++) {
        if ((seqpn = seq_prof_node_create(ch)) == NULL) {
            seq_prof_free(sp);
            return NULL;
        }
        if (seqpn->sum == 0.0) {
            seq_prof_free(sp);
            return NULL;
        }
        if (!sp->first)
            sp->first = seqpn;
        if (sp->last)
            sp->last->next = seqpn;
        sp->last = seqpn;
    }
}

```

profile.c

```

    len++;
}

sp->len = len;
check_profile_consistency(sp);
return sp;
}

/*****
void seq_prof_free (seq_prof *sp)
{
    seq_prof_node *tmp, *seqpn;
    if (!sp) return;
    seqpn = sp->first;
    while (seqpn) {
        tmp = seqpn->next;
        free(seqpn);
        seqpn = tmp;
        free(sp);
    }
}

/*****
int prepend_node (seq_prof *sp, seq_prof_node *seqpn)
{
    /* add node to the beginning of profile */
    if (!sp) {
        err("@Warning: ignoring null profile in prepend_node\n");
        return -1;
    }
    if (!seqpn) {
        err("@Warning: ignoring null profile_node in prepend_node\n");
        return -1;
    }
    if (!sp->first) {
        sp->first = sp->last = seqpn;
        sp->len = 1;
        return 0;
    }
    seqpn->next = sp->first;
    sp->first = seqpn;
    sp->len++;
    check_profile_consistency(sp);
    return 0;
}

/*****
int append_node (seq_prof *sp, seq_prof_node *seqpn)
/* add node at the end of the profile */
{
    if (!sp) {
        err("@Warning: ignoring null profile in append_node\n");
        return -1;
    }
}

```

profile.c

Listing for Adam Sartiel

Sun/Aug 9 10:33:23 1998

```

)
if (!iseqn) {
    err("Warning: ignoring null profile_node in append_node\n");
    return -1;
}

if (!isp->last) {
    sp->first = sp->last = seqpn;
    sp->len = 1;
    return 0;
}

sp->last->next = seqpn;
sp->len++;
check_profile_consistency(sp);
return 0;
}

/* insert node immediately after the given node */
int insert_node (seq_prof *sp, seq_prof_node *old_node, seq_prof_node *new_node)
{
    if (!new_node) {
        err("Warning: ignoring null new_node in insert_node\n");
        return -1;
    }
    if (old_node) {
        new_node->next = old_node->next;
        old_node->next = new_node;
        sp->len++;
    }
    else
        return prepend_node(sp, new_node);

    check_profile_consistency(sp);
    return 0;
}

/* gets the i'th node of the profile */
seq_prof_node *seqpn;
int j;

check_profile_consistency(sp);

if (i >= sp->len) {
    err("get_node : internal error, call amit/avner (index &d is bigger than pro
file's length)\n", i);
}

for (j = 0, seqpn = sp->first; j < i && seqpn != NULL; j++, seqpn = seqpn->next);
return seqpn;
}

```

profile.c

Listing for Adam Sartiel

Sun/Aug 9 10:33:23 1998

```

/*
void seq_prof_update (seq_prof *sp, int start, char *seq) {
    seq is in alignment format: for substitution it holds the y sequence in
    upper case for y-gaps a '-' sign for x-gaps the y sequence in lower case.
*/

char ch;
int start_insert = 0;
float gap_weight;

seq_prof_node *current_seqpn, *prev_seqpn, *tmp_seqpn;

if (start > 0) {
    prev_seqpn = get_node(sp, start-1);
    current_seqpn = prev_seqpn->next;
}
else {
    current_seqpn = get_node(sp, start);
    prev_seqpn = NULL;
}

while ((ch = *seq++) && current_seqpn) {
    if ((ch >= 'A' && ch <= 'Z') || ch == '-') {
        seq_prof_node *update(current_seqpn, ch);
        prev_seqpn = current_seqpn;
        current_seqpn = current_seqpn->next;
        if (start_insert)
            start_insert = 0;
    }
    else if (ch >= 'a' && ch <= 'z') {
        if (!start_insert) {
            if (prev_seqpn)
                gap_weight = prev_seqpn->sum - 1;
            else
                gap_weight = 0;
            if (current_seqpn)
                gap_weight += current_seqpn->sum;
            gap_weight /= 2;
            start_insert = 1;
        }
        tmp_seqpn = seq_prof_node_create(ch-'a'+'A');
        tmp_seqpn->gap = gap_weight;
        insert_node(sp, prev_seqpn, tmp_seqpn);
        prev_seqpn = tmp_seqpn;
    }
    else
        start_insert = 0;
}

check_profile_consistency(sp);
}

/*
seq_prof *seq_prof_split(seq_prof *sp, int offset) {
    seq_prof *spl;
    seq_prof_node *spn;
    spl = seq_prof_new();
}

```

profile.c

A-137

```

    spn = get_node(sp, offset - 1);
    if (!spn)
        return NULL;
    spl->last = sp->last;
    spl->first = spn->next;
    spl->len = sp->len - offset;

    sp->len = offset;
    sp->last = spn;
    sp->last->next=NULL;
    check_profile_consistency(spl);
    check_profile_consistency(spl);
    return spl;
}

/*****
seq_prof *seq_prof_concat(seq_prof *spl, seq_prof *sp2) {
    spl->last->next = sp2->first;
    spl->last = sp2->last;
    spl->len += sp2->len;
    check_profile_consistency(spl);
    free(sp2);
    return spl;
}

/*****
char *compute_assembly(seq_prof *sp, int ignore_gaps)
{
    int i;
    char *assembly;
    seq_prof_node *seqpn, *prev, *tmp;
    if ((assembly = (char *) malloc((sp->len+1)*sizeof(char)))==NULL)
        err("compute_assembly: memory failure trying to allocate %d for 'assembly'",
            sp->len+1);

    for(i=0, prev = NULL, seqpn = sp->first; seqpn ; seqpn = tmp) {
        tmp = seqpn->next;
        ch = seq_prof_node_char(seqpn);
        if (ch>='A' && ch<='Z') {
            prev = seqpn;
            assembly[i++] = ch;
        }
        else {
            sp->len--;
            if(!prev) {
                sp->first = tmp;
                seq_prof_node_free(seqpn);
            }
            else {
                prev->next = tmp;
                seq_prof_node_free(seqpn);
            }
        }
        assembly[i] = '\0';
        return assembly;
    }
}

```

```

/*****
char *final_compute_assembly(seq_prof *sp, int ignore_gaps,
    int *probable_mistakes, int *corrected)
{
    int i, is_error, is_corrected;
    char *assembly;
    seq_prof_node *seqpn;
    if ((assembly = (char *) malloc((sp->len+1)*sizeof(char)))==NULL)
        err("compute_assembly: memory failure trying to allocate %d for 'assembly'",
            sp->len+1);
    *probable_mistakes = 0;
    *corrected = 0;
    for(i=0, seqpn = sp->first; seqpn ; seqpn = seqpn->next) {
        err_place = i;
        ch = final_seq_prof_node_char(seqpn, &is_error, &is_corrected);
        if (is_error) (*probable_mistakes)++;
        if (is_corrected) (*corrected)++;
        if (ch>='A' && ch<='Z')
            assembly[i++] = ch;
        if (!ignore_gaps)
            assembly[i++] = ch-'A'+'A';
    }
    assembly[i] = '\0';
    return assembly;
}

/*****
void print_seq_prof_node(seq_prof_node *seqpn) {
    printf("A %.2f, C %.2f, G %.2f, T %.2f, Gap %.2f depth %d ===== %c\n", se
        qpn->a, seqpn->c, seqpn->g, seqpn->t,
            seqpn->gap, (int)(seqpn->sum+seqpn->gap), seq_prof_node_char(seqpn));
}

/*****
void print_seq_prof(seq_prof *sp) {
    seq_prof_node *seqpn;
    for (seqpn = sp->first; seqpn ; seqpn = seqpn->next)
        print_seq_prof_node(seqpn);
}

/*****
void print_seq_prof_width(seq_prof *sp)
{
    seq_prof_node *seqpn;
    int width, last_width, loc, last_loc;

    last_width=0;
    last_loc =-1;
    for (seqpn = sp->first, loc=0; ; seqpn = seqpn->next, loc++)
    {
        if (seqpn==NULL)
            width=0;
        else
            width=seqpn->sum+seqpn->gap;
        if(width != last_width)

```

```
(
    if (seqpn!=sp->first) printf("%d(%d bp) ",last_width,loc-last_loc);
    last_width=width;
    last_loc=loc;
)
if (seqpn==NULL)
    break;
printf("\n");
}

void profile__cut_head(seq_prof *sp,int cut_size)
{
    seq_prof_node *tmp;
    int i;
    if (cut_size==0) return;
    for (i=0; i<cut_size ; i++)
    {
        tmp = sp->first;
        sp->first=sp->first->next;
        seq_prof_node_free(tmp);
        if (sp->first==NULL)
            err("profile__cut_from_start : internal error, call amit/avner \n"
                "(trying to cut %d from a profile of length %d",cut_size,sp->len);
    }
    sp->len -= cut_size;
    check_profile_consistency(sp);
}

void profile__cut_tail(seq_prof *sp,int cut_size)
{
    seq_prof_node *p=sp->first,*tmp;
    int i;
    if (cut_size==0) return;
    for (i=0; i<sp->len-cut_size-1 ; i++)
    {
        if (p==NULL)
            err("profile__cut_from_end : internal error, call amit/avner \n"
                "(trying to cut %d from a profile of length %d",cut_size,sp->len);
        p = p->next;
    }
    sp->last = p;
    p = p->next;
    while (p) {
        tmp = p;
        p=p->next;
        seq_prof_node_free(tmp);
    }
    sp->last->next=NULL;
    sp->len -= cut_size;
    check_profile_consistency(sp);
}
```

```
void check_profile_consistency(seq_prof *sp)
{
    seq_prof_node *seqpn;
    int cnt;
    for (cnt=0,seqpn = sp->first; seqpn ; cnt++,seqpn = seqpn->next)
    {
        if (cnt!=sp->len)
            err("profile -- len defacto %d, len deura %d\n",cnt,sp->len);
    }
}
```



```

/* repeats - functions handling repeats in the contig
*/
/* $Log: repeats.c,v $
Revision 1.23 1998/06/29 12:09:47 eval
Include constants from custody_zones.h
Revision 1.22 1998/06/24 07:56:45 eval
Add the link parameter to the calls for connect_stitch
Revision 1.21 1998/06/18 14:42:47 ariels
Strip "#ifdef CPROF_ALIGN" and "#ifdef IMPROVE_CPROF" lines.
Revision 1.20 1998/06/15 07:55:44 eval
change the call to est_table__get_hyper_edge to est_table__get_hyper_edge_p
ath
Revision 1.19 1998/05/12 11:01:10 eval
make sure that a transcript will not be longer than MAX_TRANSC_LEN
Revision 1.18 1998/04/27 14:36:14 eval
Apparently cprof_compute_assembly returns X when there is an X in the profile
so I changed the checks in get_nrs_list to check that the pure_len of
the nrs > indep_node_len, and cancel the call for unpure_seglen
Revision 1.17 1998/04/15 10:24:21 eval
1. move functions find_nodes_types and find_all_sons to splice_graph.c
2. add a call to unify_needed_nodes in rp_mode_update_splice_graph
Revision 1.16 1998/04/14 10:22:20 eval
1. Corrected bug in check_multiple_hits in variants() which freed the
HillTops list prematurely. This caused trouble when there was more
than one repeat. (ariels)
2. Fix bug in find_cycle
3. change graph_point to hold only node id and type (end/start) and not
offset. change also all the related functions
Revision 1.15 1998/04/05 11:45:11 eval
1. Adding a boolean parameter to find_cycle which tells the function whether t
o
print the cycles or not.
2. Fixing bug in process_nrs - do not split node at middle end/start if right/
left
is too sort.
3. Fixing bug in get_nrs_list - there were unfreed nrs when left_most=NULL
4. Fixing bug in stack_pop - size-- --> --size.
Revision 1.14 1998/03/30 09:22:00 eval
1. make sure that graph_points will not be left too close to the end/beginning
of a node, but moved to the end/beginning
2. Change the field array of stack from static allocation to dynamic, and make
sure it grows when needed
3. Stop the search for transcripts when recursion is too deep (20)
Revision 1.13 1998/03/24 12:37:53 eval
Add a lot of functions for the build_consensus - does not work yet
(i.e. don't try to run in rp_mode with post_process (unless you compile
with -DONLTY_TRANS))

```

repeats.c

```

Revision 1.12 1998/03/16 06:16:57 eval
1. correcting the calls for split_node
2. removing identicals paths from the reported transcripts
Revision 1.11 1998/02/24 07:11:24 eval
Fixing bugs in build_transcripts
Revision 1.10 1998/02/23 15:00:25 eval
1. Changing process_nrs to cat the node before recursion.
2. Working on build transcripts.
Revision 1.9 1998/02/21 23:02:06 avner
changed 'check_repeat':
call HillTops with 2*cutoff, 2*bound (in both directions).
printout changed to include identity/similarity of the repeat alignment.
maximum similaty (for every direction) is calculated.
Changed 'stack_t' to 'stack_type' as the first caused confusion with lib
definition.
Revision 1.8 1998/02/19 16:09:00 eval
Putting rp_mode_build_transcripts and all its functions under CPROF_ALIGN
Revision 1.7 1998/02/19 14:25:40 eval
Adding rp_mode_build_transcripts functions
Revision 1.6 1998/02/17 09:10:22 ariels
Changed cluster_no to contig_name.
Revision 1.5 1998/02/16 14:54:56 eval
Working on build_transcripts.
Revision 1.4 1998/02/15 08:56:19 eval
Fixing bug in get_nrs_list - the profile was cut according to the HillTops in
formation
which was calculated to the computed assembly of the profile.
Revision 1.3 1998/02/11 12:57:53 avner
Added Id and ChangeLog comments.
*/
static char rcsid[] = "$Id: repeats.c,v 1.23 1998/06/29 12:09:47 eval Exp $";
#include <assert.h>
#include <string.h>
#include "cprof.h"
#include "hyper_graph.h"
#include "error.h"
#include "repeats.h"
#include "est_table.h"
#include "custody_zones.h"
extern int indep_node_len, bound, cutoff, phase2_id_bound;
extern double straight_repeat_max_sim, inverted_repeat_max_sim;
extern char p_graph_operations, p_ests, p_alignstr, contig_name[];
static int process_nrs(splice_graph *graph, no_repeat_seq *nrs,
graph_point **start, graph_point **end,
graph_point **points_to_update, char *indentation);
static no_repeat_seq *get_nrs_list(cprof_profile, int recur_shift);
static no_repeat_seq *new_nrs(cprof_profile, int start, int end);

```

repeats.c

```

static no_repeat_seq *find_nrs(splice_graph *graph, char *sequence);
static void free_nrs(no_repeat_seq *nrs);
static void free_nrs_list(no_repeat_seq **nrs);
static graph_point *graph_point_new(int node, int offset, int type);
static void push_point(graph_point **stack, graph_point *point);
static void free_point(stack(graph_point **stack));
static void print_cycle(splice_node_container **stack,
                        splice_node_container **node_cont);
static splice_node_container *dfs_stack_push(splice_node_container **stack, splice_node_container **stack);
static splice_node_container *dfs_stack_find(splice_node_container **stack, splice_node_container **node);
static int cyclic_dfs(splice_graph *graph, splice_node *node, splice_node_container **stack, int print_cycles);
static stack_type *stack_new(void);
static void print_paths(int **paths, int transc_no);
static void stack_free(stack_type *stack);
static void stack_push(stack_type *stack, int id);
static int stack_pop(stack_type *stack);
static int stack_at(stack_type *stack, int idx);
static void stack_print(stack_type *stack);
static splice_node *consensus_node_new();
static void consensus_graph_create_nodes(consensus_graph *dag, int **transcripts, int transc_no, splice_graph *graph);
static consensus_graph *consensus_graph_new();
static consensus_graph *find_good_dag(splice_graph *graph, int **transcripts, int transc_no);
static void consensus_graph_add_edge(consensus_node *source, consensus_node *target);
static consensus_node *consensus_graph_find_replica(int replica_id, int node_idx, consensus_graph *graph);
static void consensus_graph_add_transcript(consensus_graph *dag, int *path);
static int consensus_graph_is_dag(consensus_graph *graph);
static void consensus_graph_remove_transcript(consensus_graph *graph, int *path);
static void consensus_graph_remove_edge(consensus_node *source, consensus_node *target);

/*
 *
 */
/*
 *
 */
/*
 *
 */
void check_repeats_within_variants(splice_graph *graph)
{
    char *variant_seq, var_type[15];
    splice_node *variant_node;
    int i;
    int dummy;
    for (i=0; i<graph->size; i++) {
        variant_node=graph->node_list[i];
        variant_seq = cprof_compute_assembly(variant_node->profile, 0, &dummy, &dummy);

        sprintf(var_type, "variant %d ", i);
        check_repeat(variant_seq, var_type);
        free(variant_seq);
    }
}

```

```

int check_repeat(char *seq, char *type_seq)
{
    char *inv, *tmp;
    hilltop *list, *ht;
    int has_repeats = 0;

    list = Hilltops (seq, seq, STRAIGHT, 2*cutoff, 2*bound);
    for (ht = list; ht != NULL; ht = ht->next) {
        get_alignment(seq, seq, ht, BEST);
        if (abs(ht->x0 - ht->y0) > 25) {
            printf ("repeat in %s %s %3d..%3d <=> %3d..%3d\n",
                    " [id %3.0f, sim %3.0f]",
                    ht->id_percent, ht->sim_percent);
            if (ht->sim_percent > straight_repeat_max_sim)
                straight_repeat_max_sim = ht->sim_percent;
            has_repeats |= STRAIGHT_REPEAT;
        }
    }
    destroy_hilltop_list (&list);

    /* inverted */
    list = Hilltops (seq,
                    inv = subseq (seq, strlen (seq), 0),
                    INVERTED, 2*cutoff, 2*bound);

    free(inv);
    for (ht = list; ht != NULL; ht = ht->next)
    {
        get_alignment(seq, seq, ht, BEST);
        printf ("repeat in %s %s %3d..%3d <=> %3d..%3d\n",
                "[id %3.0f, sim %3.0f]",
                ht->id_percent, ht->sim_percent);
        if (ht->sim_percent > inverted_repeat_max_sim)
            inverted_repeat_max_sim = ht->sim_percent;
        has_repeats |= INVERTED_REPEAT;
    }
    destroy_hilltop_list (&list);
    return has_repeats;
}

void check_multiple_hits_in_variants(splice_graph *graph)
{
    int loc1, loc2;
    char *tmp;
    hilltop *list, *ht;

    printf ("checking for multiple hits between the %d variants\n",
            for (loc1 = 0; loc1 < graph->size-1; loc1++)
                for (loc2 = loc1+1; loc2 < graph->size; loc2++) {
                    /* straight */
                    list = Hilltops (graph->node_list[loc1]->seq, graph->node_list[
                                loc2]->seq,
                                0, 2*cutoff, 2*bound);
                    for (ht = list; ht != NULL; ht = ht->next) {
                        get_alignment(graph->node_list[loc1]->seq, graph->node_list[
                                    loc2]->seq,
                                    ht, BEST);
                        if (ht->list || ht->next) printf("MH ");
                    }
                }
    }
}

```

repeats.c

repeats.c

```

printf ("%d <--> %d %d..%d <+> %d..%d [%3.1f]\n",
        loc1.loc2, ht->xt, ht->y0, ht->yt, ht->score);
if (p_alignstr=="y") {
    printf("align string is %s\n", tmp=get_alignment_string
        (ht_graph->node_list[loc1->seq]);
    free(tmp);
}
}
destroy_hilltop_list (&list);
}

/* Find and print cycles in the graph.
 * Algorithm: uses DFS with stack - the stack contains the node from the
 * current node to the (arbitrary) root node, when a back edge is encountered
 * the cycle it closes is printed.
 */

```

```

int find_cycle(splice_graph *graph, int print_cycles)
{
    int i, found=0;
    splice_node *node;
    splice_node_container *stack = NULL;
    for(i=0; i<graph->size; i++)
        graph->node_list[i]->marked = 0;
    for(i=0; i<graph->size; i++) {
        node = graph->node_list[i];
        if ((node->marked)
            if (cyclic_dfs(graph, node, &stack, print_cycles) == 1)
                found = 1;
        }
        return found;
    }

/* The recursive DFS function.
 * node is the current visited node.
 * stack contains the list of nodes from the current to the root.
 */
int cyclic_dfs(splice_graph *graph, splice_node *node,
               splice_node_container **stack, int print_cycles)
{
    int nbr, found_cycle = 0;
    splice_node_container node_cont, *found_node_cont;
    node_cont.node = node;
    if (node->marked) {
        if ((found_node_cont = dfs_stack_find(stack, node)) != NULL) {
            found_cycle = 1;
            if (print_cycles)
                print_cycle(stack, found_node_cont);
        }
    }
}

```

repeats:c

```

else {
    node->marked = 1;
    dfs_stack_push(stack, &node_cont);
    for(nbr=0; nbr<node->out_degree; nbr++)
        if (cyclic_dfs(graph, graph->node_list[node->out_neighbors[nbr].idx], stack,
            print_cycles))
            found_cycle = 1;
    dfs_stack_pop(stack);
    return found_cycle;
}

/* Return the node container holding the received node.
 * NULL if it doesn't exist.
 */
splice_node_container *dfs_stack_find(splice_node_container **stack, splice_node
*node)
{
    splice_node_container *curr;
    for(curr = *stack; curr != NULL; curr = curr->next)
        if (curr->node == node)
            return curr;
    return NULL;
}

void dfs_stack_push(splice_node_container **stack, splice_node_container *node_co
nt)
{
    node_cont->next = *stack;
    *stack = node_cont;
}

splice_node_container * dfs_stack_pop(splice_node_container **stack)
{
    splice_node_container *node_cont;
    if (*stack == NULL)
        err("Trying to pop an empty stack in dfs_stack_pop");
    node_cont = *stack;
    *stack = (*stack)->next;
    return node_cont;
}

/* Prints the cycle which is in the stack.
 * The stack contains a path from the root node to the last visited node
 * and node_cont is a container of a node which is a neighbour of the last
 * node.
 */
static void print_cycle(splice_node_container **stack, splice_node_container *nod
e_cont)
{
    splice_node_container *curr;
    if (*stack == node_cont)
        printf("self loop in node %d\n", node_cont->node->id);
    else {
        printf("Cycle found:\n");
        printf("%d", (*stack)->node->id);
    }
}

```

repeats:c

A-142

```

for(curr=(*stack)->next; curr != node_cont->next; curr = curr->next)
    printf(" <- (%d)", curr->node->id);
    printf("\n");
}

/*=====
** rpnode_variant_graph_to_splice_graph ::
** Take the variants' graph and create the splice-graph out of it.
** We are in rp_node here, which means we use the repeat-assembly algorithm.
**
splice_graph *rpnode_variant_graph_to_splice_graph(splice_graph *variant_graph)
{
    char *variant_seq;
    int update_splice_graph_ok, variant, dummy;
    splice_node *variant_node;
    splice_graph *graph = splice_graph_new();

    printf("\n\n--- turning variants into splice-graph (REPEAT-MODE) ----\n\n");
    variant_node=variant_graph->node_list[0];
    for (variant=0; variant<variant_graph->size; variant++) {
        /* note we don't give a special treatment to node 0 in the rpnode */
        variant_node= variant_graph->node_list[variant];
        variant_seq = variant_node->seq;
        printf("%svariant %d      #LN %d\n",
              variant, splice_node___len(variant_node));
        update_splice_graph_ok = rp_node_update_splice_graph
            (graph, variant_node->profile);
        if (!update_splice_graph_ok) {
            splice_graph_free(graph);
            return NULL;
        }
        else {
            if (p_graph_operations == 'y' || p_graph_operations == 'v') {
                printf("splice-graph after taking the %d'th variant\n", variant);
                graph___print(graph);
                printf("=====\n");
            }
        }
        return graph;
    }
}

/*=====
** rp_node_update_splice_graph:
** reducing to the case when the sequence has no repeats.
** The sequences is simply broken into units with this property (generally,
** of course, there will be only one unit) The units are incorporated
** one by one, and a 'connect' operation is applied to express the stich
** between two neighboring units.
**
int rp_node_update_splice_graph(splice_graph *graph, cprof profile)

```

repeats.c

```

splice_node *node;
graph_point *start=NULL, *prev_end=NULL, *end=NULL, *points_to_update=NULL;
align_data *czm;
no_repeat_seq *nrs, *nrs_list;
int node_id, left_id, node_len, nrs_cnt=0, profile_pos=0;
int dummy;

nrs_list = nrs = get_nrs_list(profile, 0);
while(nrs != NULL)
{
    char *sequence = cprof_compute_assembly(nrs->profile, 0, &dummy, &dummy);
    printf("*****\n\n");
    printf("nrs %d #LN %d\n", nrs_cnt++, nrs->profile->len);
    if (p_estts == 'y')
        printf("%s\n", sequence);
    if (!process_nrs(graph, nrs, &start, &end, &points_to_update, ""))
        return 0;
    connect_stitch(graph, prev_end, start,
                  nrs->profile->link(profile_pos),
                  &points_to_update, "");
}

/*
if (end != NULL) {
    node_len = splice_node___len(graph->node_list(end->node));
    if (pure_seglen(graph->node_list(end->node)->profile,
                  end->offset+1, node_len-1) > indep_node_len)
    {
        split_node(graph, end->node, end->offset, &left_id, &node_id);
        update_points_list_after_split(points_to_update, end->node,
                                      left_id, node_id, end->offset, graph);
    }
}

profile_pos += nrs->profile->len;
prev_end=end;
nrs = nrs->next;
free (sequence);
}
unify_needed_nodes(graph);
if (points_to_update != NULL && points_to_update->next != NULL &&
    points_to_update->next->next != NULL)
    err("points stack contains more than 2 points at the end");
else {
    free_point_stack(&points_to_update);
}
free nrs_list(nrs_list);
return 1;
}

#define MAX_RECURS_RPMODE 30
int process_nrs(splice_graph *graph, no_repeat_seq *nrs,
               graph_point **start, graph_point **end,
               graph_point **points_to_update, char *indentation)
{
    extern char p_estts;
    align_data *middle;

```

repeats.c

```

no_repeat_seq *left,*right;
graph_point *left_end,*right_start,*middle_start,*middle_end;
splice_node *node;
char new_indentation(MAX_RECURS_RPMODE*7+1);
int dummy,left_id,node_id,start_node,end_node;
char *sequence=NULL;
int node_len;

if (strlen(indentation)/7 > MAX_RECURS_RPMODE) {
    printf("recursion dep over MAX_RECURS_RPMODE (%d)\n",MAX_RECURS_RPMODE);
    strcpy(new_indentation,indentation);
}
else {
    sprintf(new_indentation,"%s",indentation);
}

if (pure_seglen(nrs->profile,0,nrs->profile->len-1) <= indep_node_len)
    return -1; /* so that this piece will get neglected */

align_data__get_rp_czm(graph,nrs,&left,&middle,&right);

if (middle == NULL) {
    printf("%sfree\n",indentation);
    /* There is no alignment - create a new node */
    node = node_init(cprof_segment(nrs->profile,0,nrs->profile->len-1));
    if ((node->id == add_node(graph,node)) == -1)
        return 0;

    *start = graph_point_new(node->id,0,START_POINT);
    *end = graph_point_new(node->id,splice_node__len(node) - 1,END_POINT);
    push_point(points_to_update,*start);
    push_point(points_to_update,*end);
    return 1;
}

printf("%s",indentation);
align_data__print(middle);

middle_start = graph_point_new(middle->node_id,middle->start_node,START_POINT);
middle_end = graph_point_new(middle->node_id,middle->end_node,END_POINT);

start_node = middle->start_node;
end_node = middle->end_node;

push_point(points_to_update,middle_start);
push_point(points_to_update,middle_end);

if (pure_seglen(graph->node_list[middle->node_id]->profile,0,start_node-1) >
    indep_node_len) {
    split_node(graph,middle->node_id,start_node,&left_id,&node_id);
    update_points_list_after_split(*points_to_update,middle->node_id,
        left_id,node_id,start_node,graph);

    middle_start->node = node_id;
    middle_end->node = node_id;
    end_node = start_node;
    start_node = 0;

    node_len = splice_node__len(graph->node_list[middle_end->node]);
    if (pure_seglen(graph->node_list[middle_end->node]->profile,
        end_node+1,node_len-1) > indep_node_len)

```

repeats.c

```

{
    split_node(graph,middle_end->node,end_node+1,&left_id,&node_id);
    update_points_list_after_split(*points_to_update,middle_end->node,
        left_id,node_id,end_node+1,graph);
    middle_start->node = left_id;
    middle_end->node = left_id;
}

graph->node_list[middle_start->node]->need_to_update = 0;
fix_update_node_parameters(graph->node_list[middle_start->node],
    start_node,
    middle->start_est,
    middle->align_str);

update_node(graph->node_list[middle_start->node],nrs->profile);

if (left == NULL)
    *start = middle_start;
else {
    sequence = cprof_compute_assembly(left->profile,0,&dummy,&dummy);
    printf("%sLEFT #LN %d %s\n",indentation,left->profile->len,
        p_est=="y"?sequence:"");
    switch (process_nrs(graph,left_start,&left_end,points_to_update,
        new_indentation)) {
        case 0: /* failure */
            return 0;
        case 1:
            connect_stitch(graph,left_end,middle_start,
                nrs->profile->link[middle->start_est],
                points_to_update,indentation);
            break;
        case -1:
            *start = middle_start; /* was negligible, treatment here is
                equivalent to left == NULL */
            break;
    }
    free_nrs(left);
}

if (right == NULL)
    *end = middle_end;
else {
    sequence = cprof_compute_assembly(right->profile,0,&dummy,&dummy);
    printf("%sRIGHT #LN %d %s\n",indentation,right->profile->len,
        p_est=="y"?sequence:"");
    switch (process_nrs(graph,right_start,right_end,points_to_update,
        new_indentation)) {
        case 0: /* failure */
            return 0;
        case 1:
            connect_stitch(graph,middle_end,right_start,
                nrs->profile->link[middle->end_est+1],
                points_to_update,indentation);
            break;
        case -1:
            *end = middle_end; /* was negligible, treatment here is
                equivalent to right == NULL */
            break;
    }
}

```

repeats.c

```

    }
    free_nrs(right);
}
if (sequence != NULL) free (sequence);
return 1;
}

no_repeat_seq *get_nrs_list(cprof profile, int recur_shift)
{
    char *inv, *tmp;
    hilltop *list, *ht, *leftmost_repeat=NULL;
    no_repeat_seq *nrs_list=NULL, *nrs_head=NULL, *nrs_tail=NULL;
    int dummy, unpure_x0;
    char *sequence = cprof_compute_assembly(profile, 0, &dummy, &dummy);

    list = Hilltops (sequence, sequence, STRAIGHT, 2*cutoff, 2*bound);
    for (ht = list; ht != NULL; ht = ht->next)
        if (pure_seglen(profile, ht->y0, ht->x0-1) > indep_node_len &&
            (leftmost_repeat == NULL || ht->x0 < leftmost_repeat->x0))
            leftmost_repeat = ht;
    if (leftmost_repeat == NULL) {
        nrs_head = new_nrs(profile, 0, profile->len-1);
    }
    else {
        get_alignment(sequence, sequence, leftmost_repeat, BEST);
        printf("repeat: %3d. %3d <=> %3d. %3d\n",
            leftmost_repeat->x0+recur_shift, leftmost_repeat->xt+recur_shift,
            leftmost_repeat->y0+recur_shift, leftmost_repeat->yt+recur_shift,
            leftmost_repeat->idpercent, leftmost_repeat->simpercent);
        if (p_alignstr=="y") {
            printf("align string is %s\n", tmp=get_alignment_string(
                leftmost_repeat, sequence));
            free(tmp);
        }
        /*
        unpure_x0 = unpure_seglen(profile, 0, leftmost_repeat->x0);
        nrs_head = new_nrs(profile, 0, unpure_x0-1);
        nrs_tail = new_nrs(profile, unpure_x0, profile->len-1);
        nrs_list = get_nrs_list(nrs_tail->profile, recur_shift+unpure_x0);
        */
        nrs_head = new_nrs(profile, 0, leftmost_repeat->x0-1);
        nrs_tail = new_nrs(profile, leftmost_repeat->x0, profile->len-1);
        nrs_list = get_nrs_list(nrs_tail->profile, recur_shift+leftmost_repeat->x0);
        nrs_head->next = nrs_list;
        nrs_list = nrs_head;
    }
    free (sequence);
    if (nrs_tail)
        free_nrs(nrs_tail);
    destroy_hilltop_list (alist);
    return nrs_head;
}

```

repeats.c

A-145

```

int align_data___get_rp_czm(splice_graph *graph, no_repeat_seq *nrs,
    no_repeat_seq **left,
    align_data **middle, no_repeat_seq **right)
{
    int i;
    align_data *node_align_list, *graph_align_list=NULL, *ald, *first, *last, *max=NULL;
    splice_node *node;

    *middle = NULL;
    for (i=0; i<graph->size; i++) {
        /* for every node in the graph see how it interacts with the */
        /* profile of the next variant */
        node = graph->node_list[i];
        node_align_list = align_node_to_cprof (node, nrs->profile);
        if (node_align_list == NULL) continue;

        last = node_align_list;
        while (last->next)
            last = last->next;
        last->next = graph_align_list;
        graph_align_list = node_align_list;
    }
    align_data_list_identity_filter(&graph_align_list, phase2_id_bound);
    for (max=ald=graph_align_list; ald!=NULL; ald=ald->next)
        if (ald->score > max->score)
            max = ald;
    if (max == NULL)
        return NULL;
    *middle = align_data_new(max, -1, -1);
    align_data_list_free(graph_align_list);
    *left = new_nrs(nrs->profile, 0, (*middle)->start_est-1);
    *right = new_nrs(nrs->profile, (*middle)->end_est+1, nrs->profile->len-1);
}

no_repeat_seq *new_nrs(cprof profile, int start, int end)
{
    no_repeat_seq *nrs;
    char *seq;
    int seq_len = end - start + 1;

    if (seq_len <= 0) return NULL;
    nrs = (no_repeat_seq*)malloc(sizeof(no_repeat_seq));
    nrs->next = NULL;
    nrs->profile = cprof_segment(profile, start, end);
    return nrs;
}

void free_nrs(no_repeat_seq *nrs)
{
    free(nrs->profile);
    free(nrs);
}

```

repeats.c


```

void free_nrs_list(no_repeat_seq *nrs)
{
    if (nrs == NULL) return;
    free_nrs_list(nrs->next);
    free_nrs(nrs);
}

void update_points_list_after_split(graph_point * points_to_update,
int old_node,int left,int right,int offset,splice_graph *
graph)
{
    graph_point *point = points_to_update;
    while(point != NULL) {
        if (point->node == old_node) {
            if (point->type == START_POINT) {
                point->node = left;
            }
            if (point->type == END_POINT) {
                point->node = right;
            }
            point = point->next;
        }
    }

    void update_points_list_after_unify(graph_point *points_to_update,int left,int r
ight,
    {
        graph_point *point = points_to_update;
        while(point != NULL)
        {
            if (point->node == right) {
                point->node = left;
            }
            if (point->node == replaced_node) {
                point->node = right;
            }
            point = point->next;
        }
    }

    void push_point(graph_point **stack,graph_point*point)
    {
        if( *stack == NULL )
            point->next = NULL;
        else
            point->next = *stack;
        *stack = point;
    }

    int delete_point(graph_point** stack,graph_point *point)
    {
        graph_point* curr = NULL;
        if (*stack == NULL)

```

repeats.c

```

err("points stack: can't delete point - empty stack");
if (*stack == point) {
    *stack = point->next;
}
else {
    for(curr = *stack;curr != NULL && curr->next != point;curr = curr->next)
    ;
    if(curr == NULL)
        err("points stack: can't delete point - not found");
    curr->next = curr->next->next;
    free(point);
    return 1;
}

graph_point *graph_point_new(int node,int offset,int type)
{
    graph_point *new_point;
    new_point = (graph_point*)malloc(sizeof(graph_point));
    if (new_point == NULL)
        err("Can't allocate memory in graph_point_new");
    new_point->node = node;
    new_point->offset = offset;
    new_point->next = NULL;
    new_point->type = type;
    return new_point;
}

void free_point_stack(graph_point **stack)
{
    graph_point *curr;
    for(curr=*stack;curr != NULL;*stack = curr){
        curr = (*stack)->next;
        free(*stack);
    }
}

/** build transcripts */
void rp_mode_build_transcripts(splice_graph *graph,
char **transcripts[MAX_TRANSC],
int *transc_no, int **transc_map)
{
    *transc_no=0;
    build_transcripts_paths(graph,transc_map,transc_no);
    transc_paths_to_transc(graph,transcripts,transc_no,transc_map);
    print_paths(transc_map,*transc_no);
}

void build_transcripts_paths(splice_graph *graph,int **paths,int *transc_num)
{
    int i,cycle;
    hyper_edge *edge;
    stack_type *stack=NULL;
    hyper_graph *h_graph;

```

repeats.c


```

stack = stack_new();
h_graph = build_hyper_graph(graph);

hyper_graph_print(h_graph);

for(i=0; i<MAX_TRANSC; i++)
    paths[i] = NULL;

find_nodes_types(graph);

for(i=0; i<graph->size; i++)
    graph->node_list[i] -> marked = 0;
for(i=0; i<h_graph->size; i++)
    hyper_graph_get_edge(i, h_graph) -> mark = 0;

for(i=0; i<h_graph->size; i++) {
    edge = hyper_graph_get_edge(i, h_graph);
    if (hyper_edge__is_initial(edge))
        if (!find_paths(h_graph, edge, stack, paths, transc_num))
            break;
}

stack__free(stack);
hyper_graph__free(h_graph);

void transc_paths_to_transc(splice_graph *graph,
                           char *transcripts[MAX_TRANSC],
                           int *transc_no, int **paths)
{
    int i, pos;
    for(i=0; i<*transc_no; i++) {
        if (!transcripts[i] = (char*) malloc(MAX_TRANSC_LEN)) == NULL)
            err("Memory problem in transc_paths_to_transc");
        transcripts[i][0] = NULL;

        for(pos=0; paths[i][pos] != -1; pos++) {
            if (strlen(transcripts[i]) + strlen(graph->node_list[paths[i][pos]] -> seq)
                >= MAX_TRANSC_LEN) {
                record_warn("transcript %d is too long - cut at %d'th node (%d) \n",
                    i, paths[i][pos], pos);
                break;
            }
            strcat(transcripts[i], graph->node_list[paths[i][pos]] -> seq);
        }
    }

    int find_paths(hyper_graph *graph,
                   hyper_edge *edge,
                   stack_type *stack,
                   int **paths,
                   int *transc_num)
    {
        int i, nbr, recursion_base;

        int max_recursion = 20;

```

repeats.c

```

hyper_edge__mark_last_node(edge);
if (stack->size == max_recursion) {
    printf("recursion too deep (%d) - ending recursion\n", stack->size);
    record_warn("MAX_RECURSION (%d) was reached in build_transcripts",
        max_recursion);
    return 0;
}
stack_push(stack, edge->id);
#ifdef DBG
    stack__print(stack);
#endif
edge->mark++;
recursion_base = 1;
for(nbr=0; nbr<edge->out_degree; nbr++) {
    if (hyper_graph_good_edge(graph, edge->neighbors[nbr].target,
        if (!find_paths(graph, edge->neighbors[nbr].target,
            stack, paths, transc_num))
        return 0;
    recursion_base = 0;
}
if (recursion_base == 1 && (hyper_edge__is_end(edge))) {
    if (!write_path(stack, paths, transc_num, graph)) {
        printf("new path found (%d)\n", *transc_num);
        printf("Path: ");
        for(i=0; paths[*transc_num][i] != -1; i++)
            printf("%d ", paths[*transc_num][i]);
        printf("\n");
        (*transc_num)++;
    }
    edge->mark--;
    stack_pop(stack);
    return 1;
}

int write_path(stack_type *stack,
               int **paths,
               int *transc_num,
               hyper_graph *graph)
{
    int i, stack_idx, node_idx=0, path_loc, rc=0;
    int *curr;
    hyper_edge *edge;
    int *new_path;
    int exists = 0;

    if ((new_path = (int *) malloc(sizeof(int) * (path_len(stack, graph) + 1))) == NULL)
        err("memory problem in write_path");

    for (path_loc=0, stack_idx=0, stack_idx<stack->size; stack_idx++) {
        edge = hyper_graph_get_edge(stack__at(stack, stack_idx), graph);
        for (; node_idx<edge->len; node_idx++, path_loc++) {
            new_path[path_loc] = edge->nodes[node_idx];
        }
        if (stack_idx+1 < stack->size)

```

repeats.c

```

/* if this is not the last edge */
node_idx =
    hyper_graph__intersection_len(stack__at(stack, stack_idx),
    stack__at(stack, stack_idx+1), graph);
}
new_path[path_loc] = -1;

/* check if path exists already */
for(i=0; i<*transc_num; i++) {
    for(path_loc=0;
        new_path[path_loc] != -1 &&
        paths[i][path_loc] != -1 &&
        paths[i][path_loc] == new_path[path_loc];
        path_loc++) {
    }
    if( new_path[path_loc] == -1 && paths[i][path_loc] == -1 ) {
        exists = 1;
        break;
    }
}

if(exists) {
    printf("Path already exists (%d) ", i);
    for(i=0; new_path[i] != -1; i++)
        printf("%d ", new_path[i]);
    printf("\n");
    free(new_path);
}
else {
    if(*transc_num < MAX_TRANSC) {
        paths[*transc_num] = new_path;
    }
    else {
        printf("Not adding transcript to alignment list:\n");
        for(i=0; new_path[i] != -1; i++)
            printf("%d ", new_path[i]);
        printf("\n");
        free(new_path);
        return 0;
    }
}
return !exists;
}

int path_len(stack_type *stack, hyper_graph *graph)
{
    int stack_idx, node_idx=0, path_len, intersection_len=0;
    int *curr;
    hyper_edge *edge;

    for(path_len=0, stack_idx=0; stack_idx<stack->size; stack_idx++) {
        edge = hyper_graph__get_edge(stack__at(stack, stack_idx), graph);
        path_len += edge->len - intersection_len;
        if(stack_idx+1 < stack->size)
            /* if this is not the last edge */
            intersection_len = hyper_graph__intersection_len(stack__at(stack, stack_idx),
            stack__at(stack, stack_idx+1), graph);
    }
    return path_len;
}

```

repeats.c

```

}

int hyper_graph__good_edge(hyper_graph *graph, int edge_idx, stack_type *stack)
{
    int i;
    hyper_edge *curr, *edge = hyper_graph__get_edge(edge_idx, graph);
    for(i=0; i<stack->size; i++)
        printf(" ");

    /* Check if there is an edge in the stack which contains the new edge */
    for(i=stack->size-1; i>=0; i--) {
        if(stack__at(stack, i) != edge_idx &&
            hyper_graph__contains(stack__at(stack, i), edge_idx, graph)) {
            printf("not good edge (%d) - edge %d contains it\n", edge_idx, stack__at(stack, i));
            return 0;
        }
    }

    if(!hyper_edge__last_node_marked(edge)) {
        printf("good edge (%d) - not marked\n", edge_idx);
        return 1;
    }

    if(edge->mark == 0) {
        /* This the first time we use this edge so its OK */
        printf("good edge (%d) - first time\n", edge_idx);
        return 1;
    }

    /* There is a cycle - check if there exists a new edge in this cycle */
    for(i=stack->size-1; i>=0; i--) {
        curr = hyper_graph__get_edge(stack__at(stack, i), graph);
        if(hyper_edge__last_node(curr) == hyper_edge__last_node(edge))
            break;
        if(curr->mark < 2) /* if the current occurrence is the first one we can go on */
            printf("good edge (%d) - edge %d is new\n", edge_idx, curr->id);
        return 1;
    }

    if(i==0)
        err("Internal error in hyper_graph_good_edge(...), call Eval");
    printf("not good edge (%d) - no new edge in cycle\n", edge_idx);
    return 0;
}

void print_paths(int **paths, int transc_no)
{
    int i, j;
    for(i=0; i<transc_no; i++) {
        printf("path number %d\n", i);
        for(j=0; paths[i][j] != -1; j++)
            printf("%d ", paths[i][j]);
    }
}

```

repeats.c

```

    printf("\n");
}

/*
 * rp_mode_build_consensus -
 * since it could be that there is no consensus in a cyclic, we shall give
 * a consensus of part of the transcripts.
 */
int build_cyclic_graph_consensus (splice_graph *graph, char *consensus, char map
[],
    int *nodes_order_arr, int **transcripts, int t
ransc_no)
{
    int agreeing_ests, candidate_num=0;
    consensus_graph *dag;

    dag = find_good_dag(graph, transcripts, transc_no);

    rp_mode_build_consensus (dag, consensus, map, nodes_order_arr);
    return 1;
}

consensus_graph *find_good_dag(splice_graph *graph, int **transcripts, int trans
c_no)
{
    int i, est_idx;
    consensus_graph *dag;
    dag = consensus_graph__new();
    consensus_graph__create_nodes(dag, transcripts, transc_no, graph);
    for(est_idx=0; est_idx<est_table__size(); est_idx++) {
        consensus_graph__add_transcript(dag,
            est_table__get_hyper_edge_path(est_idx));
    }
}

for(i=0; i<transc_no; i++) {
    consensus_graph__add_transcript(dag, transcripts[i]);
    if(!consensus_graph__is_dag(dag))
        consensus_graph__remove_transcript(dag, transcripts[i]);
}

return dag;
}

/*
 * Find a topological order of the graph, and returns 1/0 for unique/nonunique
 * such order, or -1 for no order at all (i.e. when the graph is cyclic)
 */
int rp_mode_build_consensus (consensus_graph *graph, char *consensus, char ma
p[],
    int *nodes_order_arr)
{
    consensus_node *node;
    int marked[MAX_NODES], i;
    int candidate_node, good_candidate, num_good_candidates, ordinal, unique_order,
    base_in_map=0, base_in_node, nbr;
    bzero ((char*)marked, sizeof (marked));

```

repeats.c

```

for(i=0; i<MAX_NODES; nodes_order_arr[i++]=1);

for (ordinal = graph->size-1, unique_order=1, ordinal >= 0; ordinal-->0) {
    for (candidate_node=0, num_good_candidates=0;
        candidate_node < graph->size; candidate_node++) {
        if (marked[candidate_node]) continue;
        node = graph->node_list[candidate_node];
        for (nbr = 0;
            nbr < node->out_degree && marked[node->out_neighbors[nbr].target->id]
            nbr++)
            if (nbr==node->out_degree) { /* candidate_node is a good candidate */
                num_good_candidates++;
                good_candidate = candidate_node;
            }
        if (num_good_candidates==0)
            return -1;
        if (num_good_candidates > 1)
            unique_order=0;
        nodes_order_arr[ordinal] = good_candidate;
        marked[good_candidate] = 1;
    }

    if (graph->size > 1)
        printf ("order of nodes in the consensus is: ");
    for (ordinal = 0, consensus[0]='\0'; ordinal < graph->size; ordinal++) {
        strcat(consensus, graph->node_list[nodes_order_arr[ordinal]]->node->seq);
        if (graph->size > 1)
            printf ("%d ", nodes_order_arr[ordinal]);
        for (base_in_node=0;
            base_in_node < strlen
            (graph->node_list[nodes_order_arr[ordinal]]->node->seq);
            base_in_node++)
            map[base_in_map++] = ordinal;
    }
    printf("\n");
    map[base_in_map] = -1;
    return unique_order;
}

/** consensus_graph ***/
consensus_graph *consensus_graph__new()
{
    consensus_graph *new_graph;
    if(new_graph = (consensus_graph*)malloc(sizeof(consensus_graph))) == NULL)
        err("memory problem in consensus_graph__new()");
    new_graph->size = 0;
    new_graph->next = NULL;
    return new_graph;
}

/*
 * consensus_graph__create_nodes - for each node in the splice_graph we
 * add a node in the consensus_graph and we replicate it so that the number
 * of replication will be the max replication of that node in a transcript
 */
void consensus_graph__create_nodes(consensus_graph *dag, int **transcripts, in
t transc_no,

```

repeats.c

A-149

```

splice_graph *graph)
{
    int i, pos, node_idx;

    dag->size = graph->size;
    for(i=0; i<transc_no; i++) {
        for(pos=0; transcripts[i][pos] != -1; pos++) {
            node_idx = transcripts[i][pos];
            if(dag->node_list[node_idx] == NULL) {
                dag->node_list[node_idx] = consensus_node_new();
                dag->node_list[node_idx]-->node = graph->node_list[node_idx];
                dag->node_list[node_idx]-->id = node_idx;
            }
            else {
                if(node_occurrence(transcripts[i], pos, node_idx) > dag->node_list[node_idx]
                    x)->num_replicas) {
                    dag->size++;
                    dag->node_list[dag->size] = consensus_node_new();
                    dag->node_list[dag->size]-->node = graph->node_list[node_idx];
                    dag->node_list[dag->size]-->id = dag->size;
                    dag->node_list[dag->size]-->replica_id = dag->node_list[node_idx]-->num_replicas++;
                }
            }
        }
    }

    /*
     * node_occurrence - Returns the number of occurrence in PATH of NODE_IDX
     * from the start till POS (including path[pos])
     */
    int node_occurrence(int *path, int pos, int node_idx)
    {
        int i, occurrence=0;
        for(i=0; i<pos && path[i] != -1; i++) {
            if(path[i] == node_idx)
                occurrence++;
        }
        if(path[i] == -1)
            err("pos is bigger than path len(%d > %d", pos, i);
        return occurrence;
    }

    /*
     * node_occurrence_from_end - Returns the number of occurrence in PATH of NODE_ID
     * from the POS till the end of path (not including path[pos])
     */
    int node_occurrence_from_end(int *path, int pos, int node_idx)
    {
        int i, occurrence=0;
        for(i=pos+1; path[i] != -1; i++) {
            if(path[i] == node_idx)
                occurrence++;
        }
        return occurrence;
    }
}

```

repeats.c

```

* consensus_graph__add_transcript - Adds the edges which the received path def
  ines.
  * For example:
  * path = 1231245;
  * node list = 1.1 1.2 2.1 2.2 3 4.1 4.2 5
  * the following edge will be added:
  * 1.1->2.1, 2.1->3, 3->1.2, 1.2->2.2, 2.2->4.2, 4.1->5
  * note that when you look at 4 from the left (i.e. 2.2) it assumed to be 4.2,
  but
  * from the right (i.e. 5) it is 4.1 (maximum entropy)
  */
void consensus_graph__add_transcript(consensus_graph *dag, int *path)
{
    int i;
    consensus_node *source, *target;

    for(i=0; path[i+1] != -1; i++) {
        source = consensus_graph__find_replica(node_occurrence(path, i, path[i]), path[i], dag);
        target = consensus_graph__find_replica(dag->node_list[path[i+1]]->num_replica_id - 1, dag);
        consensus_graph__add_edge(source, target, path[i+1], dag);
    }
}

/*
 * consensus_graph__find_replica - Returns the REPLICA_ID'th replica of NODE_IDX
 */
consensus_node *consensus_graph__find_replica(int replica_id, int node_idx, consensus_graph *graph)
{
    int i;
    for(i=node_idx; i<graph->size; i++) {
        if(graph->node_list[i]->id == node_idx && graph->node_list[i]->replica_id == replica_id)
            return graph->node_list[i];
    }
    return NULL;
}

/*
 * consensus_graph__add_edge - Adds edges between SOURCE and TARGET if not exist
 */
void consensus_graph__add_edge(consensus_node *source, consensus_node *target)
{
    int i;
    for(i=0; i<source->out_degree; i++) {
        if(source->out_neighbors[i].target == target) {
            source->out_neighbors[i].width++;
            break;
        }
    }
    if(i == source->out_degree) {
        source->out_degree++;
        source->out_neighbors[source->out_degree].target = target;
        source->out_neighbors[source->out_degree].width = 1;
    }
}

```

repeats.c

A-150

```

    }
}

int consensus_graph__is_dag(consensus_graph *graph)
{
    int i, mark[MAX_NODES];
    bzero ((char*)mark, sizeof (mark));

    for(i=0; i<graph->size; i++) {
        if(!mark[i])
            if(consensus_node__mark_all_sons(graph, graph->node_list[i], mark))
                return 0;
    }
    return 1;
}

/*
 * consensus_node__mark_all_sons - marks all the sons (and sons sons ..) of NOD
 * array.
 * Returns 0 if no cycle was found 1 otherwise.
 */
int consensus_node__mark_all_sons(consensus_graph *graph, consensus_node *node, int *mark)
{
    int i;
    if(mark[node->id]) return 1;
    mark[node->id] = 1;

    for(i=0; i<node->out_degree; i++) {
        if(consensus_node__mark_all_sons(graph, node->out_neighbors[i], target, mark))
            return 1;
    }
    return 0;
}

/*
 * consensus_graph__remove_transcript - remove the edges defined by PATH
 * i.e. decrease the width of the edge, if it reaches zero remove it.
 */
void consensus_graph__remove_transcript(consensus_graph *graph, int *path)
{
    int i;
    consensus_node *source, *target;

    for(i=0; path[i+1] != -1; i++) {
        source = consensus_graph__find_replica(node_occurrence(path, i, path[i]), path, graph);
        target = consensus_graph__find_replica(graph->node_list[path[i+1]]->num_replicas -
            path[i+1], path, graph);
        consensus_graph__remove_edge(source, target);
    }

    void consensus_graph__remove_edge(consensus_node *source, consensus_node *target)
    {
        int i;
        for(i=0; i<source->out_degree; i++) {

```

repeats.c

```

    if(source->out_neighbors[i].target == target){
        source->out_neighbors[i].width--;
        break;
    }
    if(i==source->out_degree) {
        err("Removing non existing edge ( %d->%d )", source->id, target->id);
    }
    if(source->out_neighbors[i].width == 0) {
        source->out_degree--;
        source->out_neighbors[i].target = source->out_neighbors[source->out_degree].target;
        source->out_neighbors[i].width = source->out_neighbors[source->out_degree].width;
        idth;
    }
}

/** consensus_node ***/
consensus_node *consensus_node__new()
{
    consensus_node *new_node;
    if((new_node = (consensus_node*)malloc(sizeof(consensus_node))) == NULL)
        err("Memory problem in consensus_node__new()");
    new_node->out_degree = 0;
    new_node->num_replicas = 1;
    new_node->replica_id = 1;

    return new_node;
}

/** Stack ***/
stack_type *stack__new()
{
    stack_type *new_stack = (stack_type*)malloc(sizeof(stack_type));
    new_stack->size = 0;
    new_stack->array = (int*)malloc(sizeof(int)*MAX_PATH_LEN);
    new_stack->allocated_size = MAX_PATH_LEN;
    return new_stack;
}

void stack__free(stack_type *stack)
{
    free(stack->array);
    free(stack);
}

void stack_push(stack_type *stack, int id)
{
    if(stack->allocated_size == stack->size)
        if(!((stack->array = (int*)realloc(stack->array, 2*stack->allocated_size)) == NULL))
            err("Memory failure in stack_push for %d ids", 2*stack->allocated_size);
    stack->array[stack->size++] = id;
}

int stack_pop(stack_type *stack)
{
    return stack->array[--stack->size];
}

```

repeats.c

Sun Aug 9 10:33:23 1998

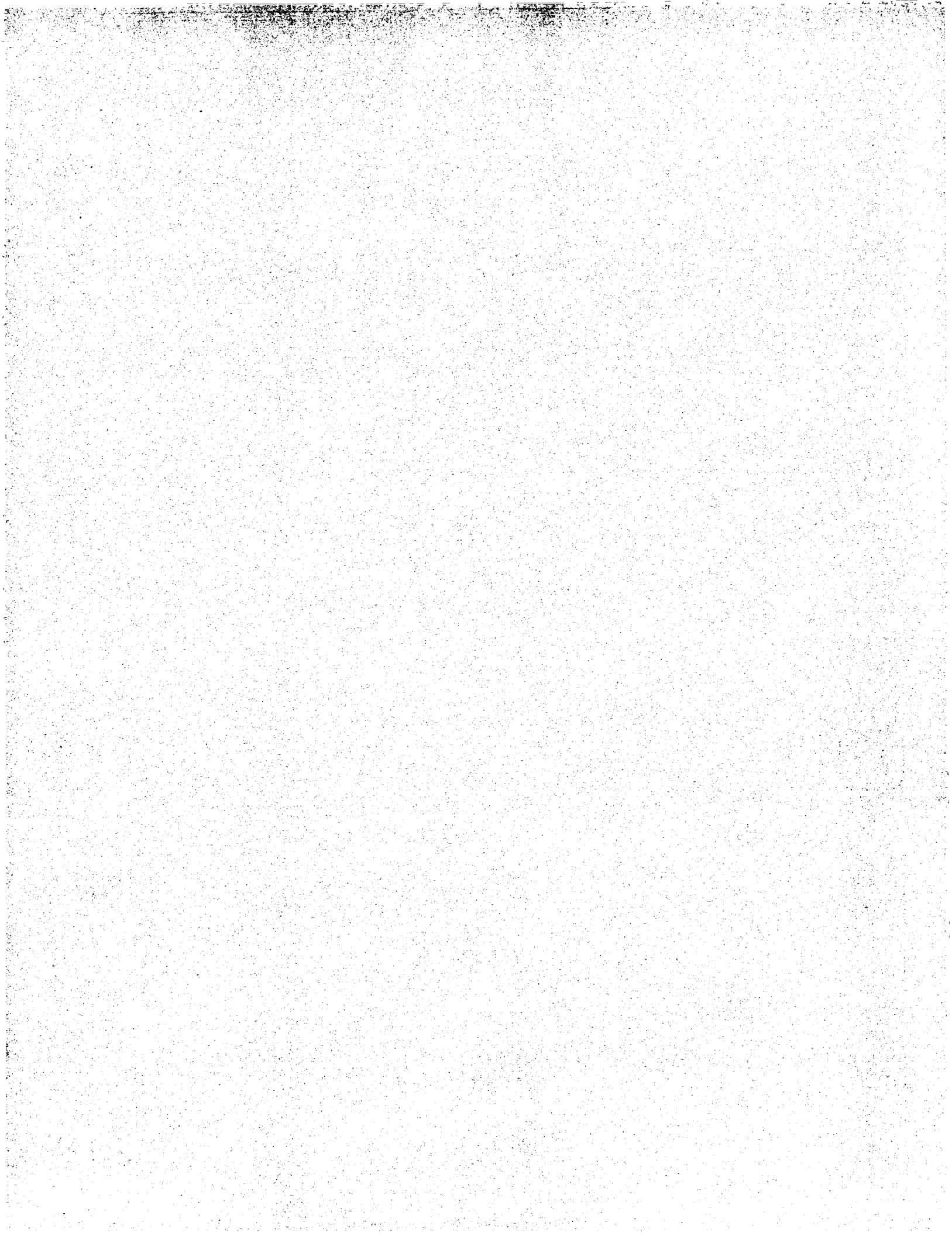
Using for Adam Santiel

A-152

```
int stack__at(stack_type *stack,int idx)
{
    assert(idx < stack->size);
    return stack->array[idx];
}

void stack__print(stack_type *stack)
{
    int i;
    for(i=0;i<stack->size;i++)
        printf(" ");
    printf("base ");
    for(i=0;i<stack->size;i++)
        printf("%d ",stack__at(stack,i));
    printf("head\n");
}
```

repeats.c



Listing for Adam Sartiell

Sun Aug 9 10:33:24 1998

```

/* Implementation of Fasta and rich-Fasta i/o functions */
/* $Log: rf.c,v $
 * Revision 1.18  1998/06/25  05:47:07  avner
 * call 'rich_fasta_post_parse_analysis' just after the sequence string has been
 * read (and not prior to it as before). This allows us to skip the length
 * field #NL.
 *
 * Revision 1.17  1998/04/26  05:49:25  avner
 * stop handling documentation. It contained a bug, and we don't use it anyway
 * (it's like appendix really).
 *
 * Revision 1.16  1998/04/25  16:26:56  avner
 * add recording of polyT existence.
 *
 * Revision 1.15  1998/04/24  23:09:36  avner
 * trim polyT as well.
 *
 * Revision 1.14  1998/04/24  12:44:23  avner
 * distinction between the two applications(assembly/flipper) added, in
 * parameter <application> to 'rich_fasta_read_seq'.
 *
 * Revision 1.13  1998/04/24  09:27:56  avner
 * add MAX_HEADER_LINE to distinguish between line size needed
 * and the size of the header. Need to be looked over again.
 *
 * Revision 1.12  1998/04/21  13:45:25  eyal
 * Change rich_fasta_seq__trim_tail and rich_fasta_seq__trim_head from
 * static to non static
 *
 * Revision 1.11  1998/04/21  09:08:40  avner
 * adding functions 'rich_fasta_seq__trim_head' and 'rich_fasta_seq__trim_tail
 * to implement the trimming we do for polyA, and (now, new) polyN. This
 * means this version will let uncleaning of quality_mark and non polyA
 * low-complexity regions only.
 *
 * Revision 1.10  1998/04/20  14:48:21  avner
 * make rfp->first_dirty = rfp->original_len when trimming polyA (caused
 * a bug in which rfp->first_dirty might be bigger than rfp->original_len).
 *
 * Revision 1.9  1998/04/19  05:25:49  avner
 * changes due to the different way in which we deal with the 'dirty' fields.
 * new semantics:
 * 'original_seq' = sequence read in the input file, minus the trimming of pol
 *
 * yA 'clean_seq' = original_seq minus quality and polyN stuff
 *
 * Revision 1.8  1998/04/06  07:39:16  avner
 * adding (the temporary probably) indication for polyA, function
 * 'check_for_polya'.
 *
 * Revision 1.7  1998/03/29  20:21:59  avner
 * continue uncleaning support work. NOTE: this version has a bug not yet
 * resolved. when working in gb<=105 and the data is >105 a segfault happens.
 *
 * Revision 1.6  1998/03/18  13:35:21  eyal
 * Add printf near malloc of original_seq

```

rf.c

Listing for Adam Sartiell

Sun Aug 9 10:33:24 1998

```

 * Revision 1.5  1998/03/17  16:45:18  avner
 * fixing typo.
 *
 * Revision 1.4  1998/03/17  16:43:29  avner
 * support the recording of tails (suspected in being dirty) of ests.
 * start supporting usage of clone-length information.
 *
 * Revision 1.3  1998/02/22  17:09:12  ariels
 * Read 3 numbers fbesrom "#SZ" field.
 *
 * Revision 1.2  1998/02/08  13:32:27  ariels
 * Correct definitions (int f()) isn't the same as int f(void!).
 * Make formatting more consistent.
 *
 * Revision 1.1  1998/02/08  13:27:49  ariels
 * Initial revision
 * */
static char resid[] = "$Id: rf.c,v 1.18 1998/06/25 05:47:07 avner Exp $";

#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <assert.h>

#include "error.h"
#include "io.h"

extern int gb_ver;
extern char dirty_tails;

static rich_fasta_seq cur_seq;
static void rich_fasta_seq_init(rich_fasta_seq_ptr rfp);
static void rich_fasta_post_parse_analysis(rich_fasta_seq_ptr rfp);
void rich_fasta_seq__trim_tail(rich_fasta_seq_ptr rfp, int cutpoint);
void rich_fasta_seq__trim_head(rich_fasta_seq_ptr rfp, int cutpoint);

static region_list_ptr region_list_add(region_list_ptr rlp,
int begin, int end)
{
    region_list_ptr tmp;

    tmp = (region_list_ptr) malloc(sizeof(region_list));
    if(!tmp) return NULL;
    tmp->begin = begin;
    tmp->end = end;
    tmp->next = rlp;
    return tmp;
}

static void region_list_free(region_list_ptr rlp)
{
    region_list_ptr tmp;

    while((tmp = rlp) != NULL) {
        rlp = rlp->next;
    }
}

```

rf.c

```

    free(tmp);
}

fasta_file rich_fasta_open(char *fname)
{
    fasta_file ff;
    ff = (fasta_file) malloc(sizeof(fasta_file_obj));
    if(!ff->fp = fopen(fname,"rt"))
        return NULL;
    rich_fasta_seq_init(&cur_seq);
    ff->last_line[0] = '\0';
    return ff;
}

void rich_fasta_close(fasta_file ff)
{
    fclose(ff->fp);
    free(ff);
}

rich_fasta_seq_ptr rich_fasta_seq_new(void)
{
    rich_fasta_seq_ptr rfp;

    rfp = (rich_fasta_seq_ptr) malloc(sizeof(rich_fasta_seq));
    if(!rfp) return NULL;
    rfp->original_seq = NULL;
    rfp->cleaned_seq = NULL;
    rfp->accession = NULL;
    rfp->id = NULL;
    rfp->name = NULL;
    rfp->clone = NULL;
    rfp->tissue = NULL;
    rfp->library = NULL;
    rfp->chromosome = NULL;
    rfp->definition = NULL;
    rfp->organism = NULL;
    rfp->cleaned_len = 0;
    rfp->original_len = 0;
    rfp->clone_len = -1;
    rfp->polyA_detected = 0;
    rfp->polyT_detected = 0;
    rfp->polyA = -1;
    rfp->polyT = -1;
    rfp->polyN = -1;
    rfp->quality_mark = -1;
    rfp->first_clean = 0;
    rfp->first_dirty = -1;
    rfp->trim5 = 0;
    rfp->trim3 = 0;
    rfp->direction = 0;
    rfp->size[0] = rfp->size[1] = rfp->size[2] = 0;
    rfp->cluster = NULL;
    rfp->meta_cluster = NULL;
    rfp->strand = 1; /* +/- 1 */
    rfp->type = NULL;
    rfp->vectors = NULL;
}

```

rf.c

```

rfp->repeats = NULL;
return rfp;
}

void rich_fasta_seq_free(rich_fasta_seq_ptr rfp)
{
    region_list_free(rfp->vectors);
    region_list_free(rfp->repeats);
    if(rfp->header) free(rfp->header);
    if(rfp->original_seq) free(rfp->original_seq);
    if(rfp->cleaned_seq) free(rfp->cleaned_seq);
    if(rfp->accession) free(rfp->accession);
    if(rfp->id) free(rfp->id);
    if(rfp->name) free(rfp->name);
    if(rfp->clone) free(rfp->clone);
    if(rfp->tissue) free(rfp->tissue);
    if(rfp->library) free(rfp->library);
    if(rfp->chromosome) free(rfp->chromosome);
    if(rfp->definition) free(rfp->definition);
    if(rfp->organism) free(rfp->organism);
    if(rfp->type) free(rfp->type);
    if(rfp->cluster) free(rfp->cluster);
    if(rfp->meta_cluster) free(rfp->meta_cluster);
    free(rfp);
}

static int field_descriptor(char *tmp)
{
    if (*tmp != '#') return 0;
    if (strcmp("AICTLDOBESVR",*(tmp+1)))
        return 1;
    return 0;
}

/*
 * parse a <line>, searching for all it's fields and load it into the <rfp>
 * structure. Note this is called both by flipper and assembly, and the
 * 'interesting' fields for the two are a bit different.
 */
static void rich_fasta_parse_args(rich_fasta_seq_ptr rfp, char *line)
{
    char *tmp, *tmp1, tmp_str[MAX_HEADER_LINE+1], short_field[5], ch;
    int be, en, tmp_end;

    /* store a copy of the original header */
    rfp->header = strdup(line);

    /* parse name */
    line++;
    sscanf(line, "%s", tmp_str);
    rfp->name = strdup(tmp_str);

    /* parse data fields */
    if ((tmp = strstr(line, "#AC")) != NULL)
    {
        sscanf(tmp+3, "%s", tmp_str);
        rfp->accession = strdup(tmp_str);
    }
}

```

rf.c

Listing for Adam Sartiel Sun Aug 9 10:33:24 1998

```

if((tmp = strstr(line, "#ID")) || (tmp = strstr(line, "#NI")))
{
    sscanf(tmp+3, "%s", tmp_str);
    rfp->id = strdup(tmp_str);
}
if((tmp = strstr(line, "#CL")) != NULL)
{
    sscanf(tmp+3, "%s", tmp_str);
    rfp->clone = strdup(tmp_str);
}
if((tmp = strstr(line, "#CH")) != NULL)
{
    sscanf(tmp+3, "%s", tmp_str);
    rfp->chromosome = strdup(tmp_str);
}
if((tmp = strstr(line, "#DR")) != NULL)
{
    sscanf(tmp+3, "%d", &(rfp->direction));
}
if((tmp = strstr(line, "#SZ")) != NULL)
{
    sscanf(tmp+3, "%d", &(rfp->size));
}
rfp->cluster=NULL;
if((tmp = strstr(line, "#CU")) != NULL)
{
    char **store = (char **) malloc(sizeof(char *) * rfp->size);
    sscanf(tmp+3, "%s", tmp_str);
    if((tmp = strstr(tmp_str, "_")) != NULL)
    {
        *store = strdup(tmp_str);
    }
    else
    {
        *store = strdup(tmp_str);
    }
}
if((tmp = strstr(line, "#CN")) != NULL)
{
    sscanf(tmp+3, "%s", tmp_str);
    if((tmp = strstr(tmp_str, "_")) != NULL)
    {
        rfp->cluster = strdup(tmp_str);
    }
    else
    {
        rfp->cluster = strdup(tmp_str);
    }
}
if(rfp->cluster == NULL)
    rfp->cluster = strdup("UNKNOWN");
if((tmp = strstr(line, "#TY")) != NULL)
{
    sscanf(tmp+3, "%s", tmp_str);
    rfp->type = strdup(tmp_str);
}
/* lengths, directions, cleaning and cloning stuff */
if((tmp = strstr(line, "#NL")) != NULL) /* it's 'new' after the vector */
{
    sscanf(tmp+3, "%d", &(rfp->original_len)); /* trimming, but for us it's */
    rfp->original_seq = (char *) malloc(rfp->original_len+1); /* original */
}
else /* will happen when we get a 'poor-fasta format' such as simulations */
    rfp->original_seq = (char *) malloc(MAX_SEQ_LEN+1); /* data */
if((tmp = strstr(line, "#BE")) != NULL)

```

r/c

Listing for Adam Sartiel Sun Aug 9 10:33:24 1998

```

    sscanf(tmp+3, "%d", &(rfp->trim5));
    if((tmp = strstr(line, "#EN")) != NULL)
    {
        int ln;
        sscanf(tmp+3, "%d", &ln);
        if((tmp = strstr(line, "#LN")) == NULL && (tmp = strstr(line, "#NL")) == NULL)
        {
            /* tmp */
            err("no LN field in %s", rfp->name);
            sscanf(tmp+3, "%d", &ln);
            rfp->trim3 = ln-en;
        }
        if((tmp = strstr(line, "#PT")) != NULL)
        {
            sscanf(tmp+3, "%d", &(rfp->polyT));
        }
        if((tmp = strstr(line, "#PA")) != NULL)
        {
            sscanf(tmp+3, "%d", &(rfp->polyA));
        }
        if((tmp = strstr(line, "#QL")) != NULL)
        {
            sscanf(tmp+3, "%d", &(rfp->quality_mark));
        }
        if((tmp = strstr(line, "#NC")) != NULL)
        {
            sscanf(tmp+3, "%d", &(rfp->polyN));
        }
        if((tmp = strstr(line, "#ST")) != NULL) /* ST shouldn't exist in flipper */
        {
            sscanf(tmp+3, "%s", short_field);
            rfp->strand = (strcmp(short_field, "+") == 0) ? 1 : -1;
        }
        if((tmp = strstr(line, "#IS")) != NULL)
        {
            sscanf(tmp+3, "%d", &(rfp->clone_len));
        }
        if((tmp = strstr(line, "#TI")) != NULL)
        {
            tmp+3;
            tmp1 = tmp;
            while((field_descriptor(tmp1) && *tmp1)
                tmp1++;
            ch = *tmp1;
            *tmp1 = 0;
            rfp->tissue = strdup(tmp+1);
            *tmp1 = ch;
        }
        if((tmp = strstr(line, "#LB")) != NULL)
        {
            tmp+3;
            tmp1 = tmp;
            while((field_descriptor(tmp1) && *tmp1)
                tmp1++;
            ch = *tmp1;
            *tmp1 = '\0';
            rfp->library = (char *) malloc(strlen(tmp) + 1);
            strcpy(rfp->library, tmp); /* = strdup(tmp); */
            *tmp1 = ch;
        }
        if((tmp = strstr(line, "#DE")) != NULL)
        {
            tmp+3;
            tmp1 = tmp;
            while((field_descriptor(tmp1) && *tmp1)
                tmp1++;
            ch = *tmp1;
            *tmp1 = 0;
            rfp->definition = strdup(tmp);
            *tmp1 = ch;
        }
        if((tmp = strstr(line, "#OS")) != NULL)
        {
            tmp+3;

```

r/c

A-155

```

tmp1 = tmp;
while (!field_descriptor(tmp1) && *tmp1)
    tmp1++;
ch = *tmp1;
*tmp1 = 0;
rfp->organism = strdup(tmp);
*tmp1 = ch;
}

while ((tmp = strstr(line, "#VE*")) != NULL) {
    sscanf(tmp+3, "%d-%d", &be, &len);
    rfp->vectors = region_list_add(rfp->vectors, be, len);
    strcpy(tmp, "XXX");
}

while ((tmp = strstr(line, "#RE*")) != NULL) {
    sscanf(tmp+3, "%d-%d", &be, &len);
    rfp->repeats = region_list_add(rfp->repeats, be, len);
    strcpy(tmp, "XXX");
}

/*
 * once data have been read, make needed analysis that consider
 * the dependencies between the different fields.
 */
static void rich_fasta_post_parse_analysis(rich_fasta_seq_ptr rfp)
{
    if (rfp->first_dirty == rfp->original_len;
        if (rfp->polyA == -1)
            rfp->polyA = rfp->original_len;
        if (rfp->quality_mark == -1)
            rfp->quality_mark = rfp->original_len;
        if (rfp->polyN == -1)
            rfp->polyN = rfp->original_len;
        if (rfp->strand == -1) {
            int tmp_trim3, tmp_polyT;
            tmp_polyT = rfp->original_len - rfp->polyA;
            rfp->polyA = rfp->original_len - rfp->polyT;
            rfp->polyT = tmp_polyT;
            rfp->quality_mark = rfp->original_len - rfp->quality_mark;
            rfp->polyN = rfp->original_len - rfp->polyN;
            tmp_trim3 = rfp->trim3;
            rfp->trim3 = rfp->trim5;
            rfp->trim5 = tmp_trim3;
            rfp->first_clean = max(rfp->quality_mark, rfp->polyN);
        }
    } else { /* straight strand */
        if (rfp->quality_mark != -1 && rfp->quality_mark < rfp->first_dirty)
            rfp->first_dirty = rfp->quality_mark;
        if (rfp->polyN != -1 && rfp->polyN < rfp->first_dirty)
            rfp->first_dirty = rfp->polyN;
    }
}

static void rich_fasta_seq_init(rich_fasta_seq_ptr rfp)
{
    region_list_free(rfp->vectors);
    region_list_free(rfp->repeats);

```

r/c

```

if (rfp->original_seq) free(rfp->original_seq);
if (rfp->cleaned_seq) free(rfp->cleaned_seq);
rfp->original_seq = NULL;
rfp->cleaned_seq = NULL;
if (rfp->accession) free(rfp->accession);
rfp->accession = NULL;
if (rfp->id) free(rfp->id);
rfp->id = NULL;
if (rfp->name) free(rfp->name);
rfp->name = NULL;
if (rfp->clone) free(rfp->clone);
rfp->clone = NULL;
if (rfp->tissue) free(rfp->tissue);
rfp->tissue = NULL;
if (rfp->library) free(rfp->library);
rfp->library = NULL;
if (rfp->chromosome) free(rfp->chromosome);
rfp->chromosome = NULL;
if (rfp->definition) free(rfp->definition);
rfp->definition = NULL;
if (rfp->organism) free(rfp->organism);
rfp->organism = NULL;
if (rfp->type) free(rfp->type);
if (rfp->cluster) free(rfp->cluster);
rfp->cleaned_len = 0;
rfp->original_len = 0;
rfp->cluster = NULL;
rfp->strand = 0; /* +/- 1 */
rfp->type = NULL;
rfp->vectors = NULL;
rfp->repeats = NULL;
}

/*
 * note - (gb_ver <= 105) means the sequence we have is already the cleaned
 * one.
 */
rich_fasta_seq_ptr rich_fasta_read_seq(fasta_file ff, int application)
{
    char *line = ff->last_line;
    FILE *fp = ff->fp;
    int rc = 0, i, j;
    rich_fasta_seq_ptr rfp = rich_fasta_seq_new();
    if (!rfp) return NULL;

    /* Read header line (if necessary) */
    if (strlen(line) == 0)
        do {
            rc = (fgetc(line, MAX_HEADER_LINE - 1, fp) == NULL);
        } while (rc && *line != '>');

    if (rc) {
        line[0] = '\0';
        return NULL;
    }

    /* parse comment line */
    rich_fasta_seq_init(rfp);
    rich_fasta_parse_args(rfp, line);
}

```

r/c

A-156

```

/* Read data */
j=0;
for(j=0;rc=(fgets(line,MAX_LINE_LEN-1,fp)==NULL);!rc&&*line!='>');
rc=(fgets(line,MAX_LINE_LEN-1,fp)==NULL){
for(i=0;i<strlen(line)-1;i++){
rfp->original_seq[i+j]=toupper(line[i]);
j+=strlen(line)-1;
rfp->original_seq[j]='\0';
}
if(j==0) return NULL;
if(rc) line[0]='\0';
if(rfp->original_len==0) /* for fabricated data like simulations */
rfp->original_len=strlen(rfp->original_seq);
else /* check consistency of redundant data */
if(rfp->original_len!=strlen(rfp->original_seq))
err("NL field does not match true length of sequence %s (%d<->%d)",
rfp->name,rfp->original_len,strlen(rfp->original_seq));
if(application==ASSEMBLY_MODE){
rich_fasta_post_parse_analysis(rfp);
if(gb_ver>105){
if(rfp->polyA<rfp->original_len){
rich_fasta_seq_trim_tail(rfp,rfp->polyA);
rfp->polyA_detected=1;
}
if(rfp->polyT>0){
rich_fasta_seq_trim_head(rfp,rfp->polyT);
rfp->polyT_detected=1;
}
if(rfp->strand==1&&rfp->polyN<rfp->original_len)
rich_fasta_seq_trim_tail(rfp,rfp->polyN);
if(rfp->strand==1&&rfp->polyN>0)
rich_fasta_seq_trim_head(rfp,rfp->polyN);
}
if(dirty_tails!='y'){
char*old_str=rfp->original_seq;
rfp->original_seq[rfp->first_dirty]='\0';
rfp->original_seq+=rfp->first_clean;
rfp->original_seq=strdup(rfp->original_seq);
free(old_str);
rfp->trim5+=rfp->first_clean;
rfp->trim3+=rfp->original_len-rfp->first_dirty;
rfp->original_len=(rfp->first_clean+rfp->original_len-rfp->first_dir
ty);
}
}
/* printf("seq %s, original_len=%d, first_clean=%d,first_dirty=%d,"
"trim5=%d,trim3=%d\n",rfp->name,rfp->original_len,rfp->first_clean,
rfp->first_dirty,rfp->trim5,rfp->trim3); */
return rfp;
}

int read_next_rich_fasta_seq(fasta_file ff)
{
char*line=ff->last_line;
FILE*fp=ff->fp;

```

rf.c

```

int rc,i,j;

/* Read header line (if necessary) */
if(strlen(line)==0){
for(rc=0;rc==0;){
rc=(fgets(line,MAX_HEADER_LINE-1,fp)==NULL);
if(line[0]!='>') break;
}
if(rc){
line[0]='\0';
return rc;
}
}

/* parse comment line */
rich_fasta_seq_init(&cur_seq);
rich_fasta_parse_args(&cur_seq,line);

/* Read data */
for(rc=j=0;!rc;){
rc=(fgets(line,MAX_LINE_LEN-1,fp)==NULL);
if(rc&&j==0) return rc;
if(line[0]!='>') rc break;
for(i=0;i<strlen(line)-1;i++){
cur_seq.original_seq[i+j]=toupper(line[i]);
j+=strlen(line)-1;
cur_seq.original_seq[j]='\0';
}
if(rc)
line[0]='\0';
return 0;
}

char*rich_fasta_name(void)
{ return cur_seq.name; }

char*rich_fasta_accession(void)
{ return cur_seq.accession; }

char*rich_fasta_id(void)
{ return cur_seq.id; }

char*rich_fasta_clone(void)
{ return cur_seq.clone; }

int fasta_read_seq(fasta_file ff, char*name, char*comment,
int*len, char*data)
{
char*line=ff->last_line;
FILE*fp=ff->fp;
int rc,i,j;

/* Read header line (if necessary) */
if(strlen(line)==0){
for(rc=0;rc==0;){
rc=(fgets(line,MAX_HEADER_LINE-1,fp)==NULL);
if(line[0]!='>') break;
}
if(rc){
line[0]='\0';
return rc;
}
}

/* copy comment */
strncpy(comment,line,MAX_LINE_LEN-1);
comment[strlen(comment)-1]='\0';

```

rf.c

A-157

```

if (sscanf(line, "%s %s", name) != 1)
    err("bad header line: \"%s\\n\", line);

/* Read data */
for (rc = j = 0; !rc;) {
    rc = (fgets(line, MAX_LINE_LEN-1, fp) == NULL);
    if (rc && j == 0) return (1);
    if (line[0] == '>' || rc) break;
    for (i = 0; i < strlen(line)-1; i++)
        data[i+j] = toupper(line[i]);
    j += strlen(line)-1;
    data[j] = '\0';
}
*len = strlen(data);
if (rc)
    line[0] = '\0';
return (0);
}

void rich_fasta_write(FILE *fp, rich_fasta_seq_ptr rfp)
{
    char header[MAX_HEADER_LEN];
    char *tmp, *end;
    region_list_ptr rlp;

    if (!rfp || !rfp->name) return;
    printf(header, ">%s", rfp->name);
    if (rfp->cluster)
        printf(header, "%s %s", header, rfp->cluster);
    if (rfp->meta_cluster)
        printf(header, "%s %s", header, rfp->meta_cluster);
    if (rfp->size[0] || rfp->size[1] || rfp->size[2])
        printf(header, "%s %s", header, rfp->size[0] || rfp->size[1] || rfp->size[2]);
    if (rfp->cleaned_len)
        printf(header, "%s %s", header, rfp->cleaned_len);
    if (rfp->original_len)
        printf(header, "%s %s", header, rfp->original_len);
    if (rfp->accession)
        printf(header, "%s %s", header, rfp->accession);
    if (rfp->id)
        printf(header, "%s %s", header, rfp->id);
    if (rfp->clone)
        printf(header, "%s %s", header, rfp->clone);
    if (rfp->library)
        printf(header, "%s %s", header, rfp->library);
    if (rfp->chromosome)
        printf(header, "%s %s", header, rfp->chromosome);
    if (rfp->definition)
        printf(header, "%s %s", header, rfp->definition);
    if (rfp->organism)
        printf(header, "%s %s", header, rfp->organism);
    if (rfp->strand)
        printf(header, "%s %s", header, rfp->strand);
    if (rfp->type)
        printf(header, "%s %s", header, rfp->type);
    for (rlp = rfp->vectors; rlp != NULL; rlp = rlp->next)
        printf(header, "%s %s", header, rfp->begin, rfp->end);
}

```

r/c

```

for (rlp = rfp->repeats; rlp != NULL; rlp = rlp->next)
    printf(header, "%s RE %d-%d", header, rlp->begin, rlp->end);

printf(fp, "%s\\n", header);

for (tmp = rfp->original_seq; tmp != end; tmp += 60)
    printf(fp, "%60s\\n", tmp);

}

void write_sequence(FILE *fp, char *s)
{
    char *tmp, *end = strchr(s, 0);
    for (tmp = s; tmp != end; tmp += 60)
        printf(fp, "%60s\\n", tmp);
}

void rich_fasta_seq__trim_tail(rich_fasta_seq_ptr rfp, int cutpoint)
{
    rfp->original_seq[cutpoint] = '\0';
    rfp->trim3 += rfp->original_len - cutpoint;
    rfp->original_len = cutpoint;
    /* some location fields can exceed len after trimming. they should equal it */
    if (rfp->first_dirty > rfp->original_len)
        rfp->first_dirty = rfp->original_len;
    if (rfp->polyN > rfp->original_len)
        rfp->polyN = rfp->original_len;
    if (rfp->polyA > rfp->original_len)
        rfp->polyA = rfp->original_len;
    if (rfp->quality_mark > rfp->original_len)
        rfp->quality_mark = rfp->original_len;
    assert(cutpoint >= rfp->first_clean && cutpoint >= rfp->polyT);
}

void rich_fasta_seq__trim_head(rich_fasta_seq_ptr rfp, int cutpoint)
{
    char *tmp = strdup(rfp->original_seq+cutpoint);
    free(rfp->original_seq);
    rfp->original_seq = tmp;
    rfp->original_len -= cutpoint;
    rfp->trim5 += cutpoint;
    /* shift all location fields accordingly */
    rfp->first_dirty -= cutpoint;
    rfp->first_clean -= cutpoint;
    rfp->quality_mark -= cutpoint;
    rfp->polyA -= cutpoint;
    rfp->polyT -= cutpoint;
    rfp->polyN -= cutpoint;
    /* some of the locations might be negative due to the shift. anull them */
    if (rfp->first_clean < 0) rfp->first_clean = 0;
    if (rfp->quality_mark < 0) rfp->quality_mark = 0;
    if (rfp->polyN < 0) rfp->polyN = 0;
    if (rfp->polyT < 0) rfp->polyT = 0;
    assert(rfp->first_dirty >= 0);
}

```

r/c

```

/*
 * $Log: rules.c,v $
 * Revision 1.13 1998/07/03 14:28:11 avner
 * fix bug in some rules-usage message.
 *
 * Revision 1.12 1998/06/29 12:10:14 eyal
 * Include constants from custody_zones.h
 *
 * Revision 1.11 1998/06/29 11:35:03 avner
 * stop "include"ing profile.h
 *
 * Revision 1.10 1998/06/24 07:55:26 eyal
 * 1. Add the link parameter to the call for add_edge.
 * 2. changes I took from avner.
 *
 * Revision 1.9 1998/06/18 14:48:02 avner
 * unexplained /* missing was inserted.
 *
 * Revision 1.8 1998/06/16 21:01:32 avner
 * 1. adding messages for sake of fuzziness study (RULE(...)).
 * 2. getting rid of #ifdef CPROF_ALIGN sections.
 *
 * Revision 1.7 1998/02/14 17:24:32 avner
 * Fix length calculations of tails of alignment, for the sake of consistent
 * handling of 'almost surely gaps' in locations of the profile ('X').
 *
 * Revision 1.6 1998/01/27 20:59:51 avner
 * Fixing a bug in 'apply_transformation_B': when the situation is
 * node = -----
 * seq = -----
 * that is the 'overhang-tail' in the sequence is small, and we want
 * to throw it away.
 * So far we did not update the node by the alignment data, and this is of
 * course wrong. We are correcting it here by a call to
 * 'fix_update_node_parameters' in that event.
 *
 * Revision 1.5 1998/01/18 08:58:34 ariels
 * Removed superfluous (and incorrectly spelt) "extern" declaration.
 *
 * Revision 1.4 1997/12/14 22:40:29 avner
 * dumping all previous 'ifdef's for penalty handling. It seems quite
 * obsolete now.
 *
 * Revision 1.3 1997/12/14 15:32:55 ariels
 * Corrected omission in first_node (used to coredump when called with
 * the cprof2cprof routines).
 *
 * Revision 1.2 1997/11/30 08:04:32 ariels
 * Added Changelog comment and static resid string.
 */

static char rcsid[] = "$Id: rules.c,v 1.13 1998/07/03 14:28:11 avner Exp $";

#include "splice_graph.h"
#include "custody_zones.h"
extern int indep_node_len, phase_no;
extern char p_graph_operations;

void rules_usage_message(int phase_no, int rule_no, char *msg_str, int len);

```

rules.c

```

#define DONT_MIND 65536
#define sqr(x) (x*x)

int apply_transformation_A(splice_graph *graph, align_data *czm,
                           align_data *west_bank, align_data *east_bank,
                           cprof entering_profile)
/*
 * *****
 * transformation A ---[---] ----FREE_SEG-----
 * *****
 */
{
    int tail_len, pure_tail_len, east_len, pure_east_len, west_idx, left_part,
    right_part=DUMMY;
    splice_node *node;

    west_idx = west_bank->node_id;

    fix_update_node_parameters(graph->node_list[west_idx],
                              west_bank->start_node,
                              west_bank->start_est,
                              west_bank->align_str);

    pure_east_len = pure_seglen(entering_profile, east_bank->start_est,
                              east_bank->end_est);
    pure_tail_len = pure_seglen(graph->node_list[west_idx]->profile,
                              west_bank->end_node+1,
                              splice_node__len
                              (graph->node_list[west_idx])-1);

    east_len = seglen(east_bank);
    if (west_bank->suffix && splice_graph__out_deg(graph, west_idx)==0)
    {
        /* subcase EASY */
        tail_len = splice_node__len(graph->node_list[west_idx])-
            1-west_bank->end_node;
        if (pure_east_len > pure_tail_len) {
            rules_usage_message(phase_no, 1, "A cut node", pure_tail_len);
            extend_node(graph, west_idx, entering_profile,
                      east_bank->start_est, east_bank->end_est,
                      tail_len, RIGHT_EXTENSION);
            align_data__assign(czm, west_idx, west_bank->end_node+1,
                              west_bank->end_node + seglen(czm), 0, 1);
        }
        /* delicate logic matters */
        /* : the last operation is not quite right */
    }
    else {
        rules_usage_message(phase_no, 1, "A cut est", pure_east_len);
        align_data__assign(czm, west_idx, west_bank->start_node,
                          west_bank->end_node, 0, 1);
    }
}

else {
    if (pure_east_len < indep_node_len) {
        /* don't complicate the graph unless meaningful segment is there */
        rules_usage_message(phase_no, 2, "A", pure_east_len);
        align_data__assign(czm, west_idx, west_bank->start_node,
                          west_bank->end_node, west_bank->prefix,
                          west_bank->suffix);
    }
    else {
        node = node_init(cprof, segment(entering_profile,

```

rules.c


```

        east_bank->start_est,
        east_bank->end_est));
    return 0;

    if (west_bank->suffix) { /* subcase INTERMEDIATE */
        rules_usage_message(phase_no, 3, "A", pure_tail_len);
        add_edge(graph, west_idx, node->id, node->profile->link(0));
    }
    else { /* subcase DIFFICULT */
        if (!split_node(graph, west_idx, west_bank->end_node + 1,
            &left_part, &right_part))
            return 0;
        add_edge(graph, left_part, node->id, node->profile->link(0));
        update_replace_czm_of_same_node(graph, czm, west_idx, right_part,
            west_bank->end_node+1, DONT_MIND);
    }
    align_data__assign(czm, node->id, 0, seglen(czm)-1, 1, 1); /* */
}
return 1;
}

int apply_transformation_B(splice_graph *graph, align_data *czm,
    align_data *west_bank, align_data *east_bank,
    cprof entering_profile)
{
    /* *****
    * transformation B ----FREE SEG-----|---1---
    * *****
    */
    int head_len, pure_head_len, west_len, pure_west_len,
    east_idx, left_part, right_part=DUMMY;
    splice_node *node;

    east_idx = east_bank->node_id;
    head_len = east_bank->start_node;
    west_len = seglen(west_bank);
    pure_head_len = pure_seglen(graph->node_list[east_idx]->profile,
        0, east_bank->start_node-1);
    pure_west_len = pure_seglen(entering_profile, west_bank->start_est,
        west_bank->end_est);

    if (east_bank->prefix && splice_graph__in_deg(graph, east_idx)==0)
    { /* subcase EASY */
        if (pure_west_len > pure_head_len) {
            rules_usage_message(phase_no, 1, "B cut node", pure_head_len);
            extend_node(graph, east_idx, entering_profile,
                west_bank->start_est, west_bank->end_est, head_len,
                LEFT_EXTENSION);
            czm__shift_right_of_same_node(czm, east_bank->node_id,
                west_len-head_len);
        }

        fix_update_node_parameters(graph->node_list[east_idx],
            (pure_west_len > pure_head_len) ?
                west_len:head_len,
                east_bank->start_est,
                east_bank->align_str);
    }
}

```

```

    }
    else {
        if (pure_west_len < indep_node_len) {
            /* don't complicate the graph unless meaningful segment is there */
            /* But do make sure an updating of the node will be performed */
            rules_usage_message(phase_no, 1, "B", pure_west_len);
            fix_update_node_parameters(graph->node_list[east_idx],
                east_bank->start_node,
                east_bank->start_est,
                east_bank->align_str);
        }
        else {
            node = node_init(cprof_segment(entering_profile,
                west_bank->start_est,
                west_bank->end_est));
            if ((node->id = add_node(graph, node)) == -1)
                return 0;

            if (east_bank->prefix) {
                rules_usage_message(phase_no, 3, "A", pure_head_len);
                add_edge(graph, node->id, east_idx,
                    node->profile->link(node->profile->len));
                fix_update_node_parameters(graph->node_list[east_idx],
                    east_bank->start_node,
                    east_bank->start_est,
                    east_bank->align_str);
            }
            else {
                if (!split_node(graph, east_idx,
                    east_bank->start_node,
                    &left_part, &right_part))
                    return 0;
                add_edge(graph, node->id, right_part,
                    node->profile->link(node->profile->len));
                fix_update_node_parameters(graph->node_list[right_part],
                    0, east_bank->start_est,
                    east_bank->align_str);

                update_replace_czm_of_same_node(graph, czm, east_idx, right_part,
                    east_bank->start_node,
                    east_bank->start_est);
            }
        }
        return 1;
    }

    int apply_transformation_C(splice_graph *graph, align_data *czm,
        align_data *west_bank, align_data *east_bank,
        cprof entering_profile)
    {
        /* *****
        * transformation C ----1-----|-----2-----
        * *****
        */
        int gap, west_idx, east_idx, left_west, right_west, left_east, east_len;
    }
}

```

A-161

```

        cprof entering_profile,
        int more_than_one_segment)

    {
        splice_node *node;
        int last_idx;

        last_idx = last_czm->node_id;
        if (!more_than_one_segment) {
            if (last_idx >= 0)
                fix_update_node_parameters(graph->node_list[last_idx],
                                           last_czm->start_node,
                                           last_czm->start_est,
                                           last_czm->align_str);
        }
        else {
            node = node_init(cprof_segment(entering_profile,
                                           last_czm->start_node,
                                           last_czm->start_est,
                                           last_czm->end_est));
            if ((node->id = add_node(graph,node)) == -1)
                return 0;
            if (p_graph_operations == 'y' && phase_no == 2 && node->id > 0)
                || p_graph_operations == 'v'
                printf("opening a new component\n");
        }
        return 1;
    }

    int first_node(splice_graph *graph, cprof prof)
    {
        splice_node *node = node_new();
        node->profile = cprof_dup(prof);
        if ((node->id = add_node(graph,node)) == -1)
            return 0;
        return 1;
    }

    void rules_usage_message(int phase_no, int rule_no, char *msg_str, int len)
    {
        if (len > 0)
            printf("%d rule %d %s (%d)\n", phase_no, rule_no, msg_str, len);
    }

```

```

        west_idx = west_bank->node_id;
        east_idx = east_bank->node_id;
        east_len = seglen(east_bank);
        /* deal with small intersection of east and west banks */
        if ((gap = east_bank->start_est - 1 - west_bank->end_est) < 0) {
            /* give up' on the tail of the western alignment */
            rules_usage_message(phase_no, 4, "C small intersection", -gap);
            czm_shorten_start(east_bank, 0, -gap);
            if (!pure_seglen(entering_profile, east_bank->start_est, east_bank->end_est))
                return 0;
        }
        ken < indep_node_len) { /* the remaining alignment is now too short to be ta
            rules_usage_message(phase_no, 5, "C back to A", 0);
            east_bank->node_id = FREE_SEG;
            return apply_transformation_A(graph, czm, west_bank, east_bank,
                                         entering_profile);
        }
        if (!west_bank->suffix) {
            if (!split_node(graph, west_idx, west_bank->end_node + 1,
                           &left_west, &right_west))
                return 0;
            update_replace_czm_of_same_node(graph, czm, west_idx, right_west,
                                           west_bank->end_node+1,
                                           west_bank->end_node+1);
        }
        if (west_idx == east_idx)
            east_idx = right_west;
        else
            left_west = west_idx; /* later we add an edge left_west-->right_east */
        fix_update_node_parameters(graph->node_list[left_west], west_bank->start_node,
                                   west_bank->start_est, west_bank->align_str);
        if (!east_bank->prefix) {
            if (!split_node(graph, east_idx, east_bank->start_node,
                           &left_east, &right_east))
                return 0;
            fix_update_node_parameters(graph->node_list[right_east], 0,
                                       east_bank->start_est, east_bank->align_str);
            update_replace_czm_of_same_node(graph, czm, east_idx, right_east,
                                           east_bank->start_node,
                                           east_bank->start_node);
        }
        else {
            right_east = east_idx;
            fix_update_node_parameters(graph->node_list[right_east],
                                       east_bank->start_node, east_bank->start_est,
                                       east_bank->align_str);
        }
        rules_usage_message(phase_no, 6, "C floating gap", gap);
        add_edge(graph, left_west, right_east,
                 entering_profile->link[west_bank->end_est]);
        return 1;
    }

```

int apply_transformation_single(splice_graph *graph, align_data *last_czm,


```

/*
 * A (slow) bubblesorting replacement for qsort, intended to assure
 * identical results across all platforms.
 */
/*
 * $Log: sort.c,v $
 * Revision 1.1 1998/04/28 09:44:06 ariels
 * Initial revision
 */

static char resid[] = "$Id: sort.c,v 1.1 1998/04/28 09:44:06 ariels Exp $";

#include <stdlib.h>
#include <stdio.h>

void bsort(void *vbase, size_t nel, size_t width,
           int (*cmp)(const void *, const void *))
{
    int i, j;
    short swapped;
    char *tmp, *base=(char*)vbase;

    if ((tmp = malloc(width)) == NULL) {
        fprintf(stderr, "bsort: failed to allocate %d for temporary storage\n",
                width);
        exit(1);
    }

    for(i=0; i<nel; i++) {
        for(j=0, swapped=0; j<nel-i-1; j++)
            if (cmp(base + width*j, base + width*(j+1)) > 0) {
                swapped = 1;
                memcpy(tmp, base + width*j, width);
                memcpy(base + width*j, base + width*(j+1), width);
                memcpy(base + width*(j+1), tmp, width);
            }
        if (!swapped)
            break;
    }
}

#ifdef TEST
#include <stdlib.h>
#include <stdio.h>
#define N 100

static int cmp_int(void const *a, void const *b)
{
    return (*(int*)a - *(int*)b);
}

main()
{
    int buf[N];
    short flag;

```

sort.c

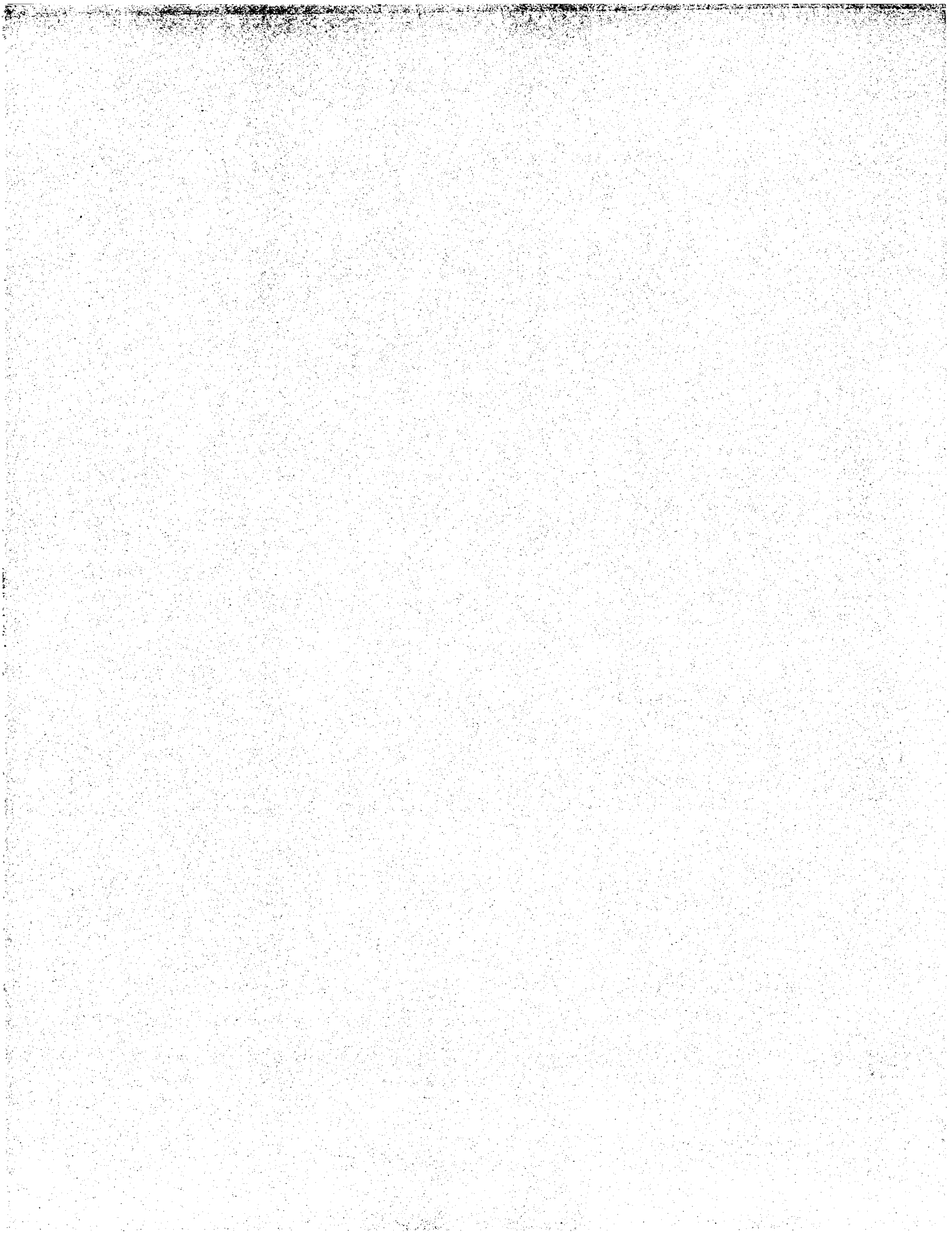
```

    int i;
    for(i=0; i<N; i++)
        buf[i] = rand();
    bsort(buf, N, sizeof(i), cmp_int);
    for(i=0, flag=0; i<N; i++)
        printf("%d: %d\n", i, buf[i]);
    if (flag) {
        printf(">>> ERROR! <<\n");
        return(1);
    }
    return(0);
}
#endif

```

sort.c

A-162



Sun Aug 9 10:33:24 1998

Listing for Adam Sartiell

```

/* implementation of some general purpose functions dealing with graph */
/*
 * $Log: splice_graph.c,v $
 * Revision 1.42  1998/07/01  13:44:32  eval
 * Go around bug of inconsistency variant_depth
 *
 * Revision 1.41  1998/07/01  06:58:45  eval
 * 1. Fix bug in splice_graph__find_connective_components
 * 2. Use cprof_ids in splice_node__print to print the variant ests
 *
 * Revision 1.40  1998/06/29  11:55:14  avner
 * change splice_node__print including the addition of numbers of
 * sequences belongin to a variant.
 *
 * Revision 1.39  1998/06/24  07:58:49  eval
 * Add the parametr link to connect_nodes and add_edge and update the width
 * field of the node accordingly.
 *
 * Revision 1.38  1998/06/24  07:24:00  eval
 * 1. create and ad a call to cprof_align_to_align_data.
 * 2. fix a bug in extend_node - the link of the profile wasn't right
 *
 * Revision 1.37  1998/06/18  14:42:47  ariels
 * Strip "#ifdef CPROF_ALIGN" and "#ifdef IMPROVE_CPROF" lines.
 *
 * Revision 1.36  1998/06/14  05:52:08  eval
 * 1. Add functions cut_est_head cut_est_tail.
 * 2. Add calls for functions cut_est_head cut_est_tail in cut_dirty_overhangs
 *
 * Revision 1.35  1998/05/11  14:39:59  eval
 * Add functions graph__print_cprofs and splice_node__print_cprof
 *
 * Revision 1.34  1998/05/04  12:39:05  eval
 * add function splice_graph__find_connective_components
 *
 * Revision 1.33  1998/04/28  13:00:50  eval
 * Remove the call to unify_nodes in connect_stitch
 *
 * Revision 1.32  1998/04/25  10:43:42  eval
 * Change 'int dirty_tails' to 'char dirty_tails'
 *
 * Revision 1.31  1998/04/23  17:44:41  avner
 * add a mechanism to avoid opening of new variants due to uncleaning.
 * We do not open a variant unless we know that even in the 'clean'
 * world it would have been formed.
 * For that the functions 'overlap_score' and 'get_best_variant' were duplicated
 * to both cases, and a function 'cut_dirty_overhangs' which does the whole
 * cutting manipulations.
 *
 * Revision 1.30  1998/04/21  09:19:37  eval
 * 1. remove cprof__get_dirty_info
 * 2. use cprof_first_clean and cprof_last_clean
 *
 * Revision 1.29  1998/04/20  14:49:44  eval
 * 1. Add function cprof__get_dirty_info - this function shouldn't be here
 * it should be in cprof.c
 * 2. Add an information print on the best alignment which is not full overlapp
 *
 * Revision 1.28  1998/04/19  07:37:21  ariels

```

splice_graph.c

Sun Aug 9 10:33:24 1998

Listing for Adam Sartiell

```

* Put #ifdef around max() in overlap_score (rather than the other was
* around), since max() is sometimes a macro (notably gcc on Sun), and
* conditional arguments were causing problems. I'm not sure if this is
* a preprocessor bug or a conformance bug on our part, but this works
* and is cleaner anyway.
 *
 * Revision 1.27  1998/04/15  10:20:33  eval
 * 1. add initialization to best_variant in get_best_variant
 * 2. add functions:
 *    splice_graph__is_initial_node
 *    splice_graph__is_ending_node
 *    find_nodes_types
 * 3. change unify_needed_nodes so it will not unify ending/initial nodes
 *    when it shouldn't
 * 4. change splice_graph__num_initial/terminals to use is_initial/ending_node
 *
 * Revision 1.26  1998/04/13  05:49:46  avner
 * The last fix was bad because instead of copying the lines I left the original
 * place without them.
 *
 * Revision 1.25  1998/04/12  15:33:29  avner
 * complete bug fix of Revision 1.23. remove the appearances of the deleted node
 * from the neighboring lists of the nodes that were directed to it, even when
 * the deleted node is the last (which was not done in 1.23).
 *
 * Revision 1.24  1998/04/12  06:35:12  eval
 * Changing connect_stitch so that it will not try to split nodes
 * since graph_points are always at the edn/start of nodes.
 *
 * Revision 1.23  1998/04/06  07:46:00  avner
 * fix bug in 'delete_node'. It didn't use to remove the appearances of the
 * deleted node from the neighboring lists of the nodes that were directed to
 * it. This was fine when we only dealt with 'unify_node', but for the general
 * case of node-deletion an extra call for 'replace_occurrences_in_neighbors_list'
 * was added.
 *
 * Revision 1.22  1998/03/06  05:33:39  eval
 * 1. Changing connect_stitch to use pure_seglen.
 * 2. Fixing the calls to split_node in connect_stitch.
 *
 * Revision 1.21  1998/02/22  17:08:18  ariels
 * Correct formatting and spelling of "stitch".
 *
 * Revision 1.20  1998/02/21  23:32:06  avner
 * 'are_neighbors' is now 'splice_graph__is_directed_edge'.
 * added 'splice_graph__is_undirected_edge', 'splice_graph__is_connected'.
 * delete commented out function 'enumerate_paths_from_node'.
 *
 * Revision 1.19  1998/02/17  16:15:34  ariels
 * Fix name of global variable cluster_no (to contig_name), since it was
 * neither CLUSTER nor NO (number)...
 *
 * Revision 1.18  1998/02/15  22:28:29  avner
 * Detect an edge formed with nodes that existed already in the old graph.
 * This means it is not due to a broken overlap. Might (doubtly) give us
 * some hint for novelty.
 *
 * Revision 1.17  1998/02/14  17:23:15  avner
 * Fix length calculations of tails of alignment, for the sake of concise

```

splice_graph.c

A-163

```

* handling of 'almost surely gaps' in locations of the profile ('X').
* Revision 1.16 1998/02/10 11:11:07 avner
* use 'indep_node_len' instead of 'olap_fuzziness' which is gone forever.
* Revision 1.15 1998/02/01 22:24:25 avner
* Fixing uninitialized variable problem 'get_best_variant'.
* Revision 1.14 1998/02/01 07:03:03 eval
* Adding function align_data_list_identity_filter.
* Fixing function best_overlap to return the best overlap.
* Revision 1.13 1998/01/21 07:54:25 ariels
* Put err() in scope of prototype (add #include "error.h")
* Revision 1.12 1998/01/14 15:24:34 eval
* Adding a call to function calc_id_sim
* Revision 1.11 1998/01/12 11:48:41 eval
* Adding casting in bzero call
* Revision 1.10 1998/01/11 14:28:55 eval
* Fix support for ident and similarity percent in align_data
* Revision 1.9 1998/01/11 14:20:02 eval
* Adding support for ident and similarity percent in align_data
* Revision 1.8 1998/01/08 13:43:48 ariels
* Use ***correct*** routines for varargs error reporting (previously worked
* fortuitously on alphas because of compiler quirks).
* No longer need to pass prm to cprof_alignment.
* Revision 1.7 1997/12/31 18:13:30 avner
* Print message for 'connect_stitch'.
* Revision 1.6 1997/12/29 16:03:59 eval
* Adding function connect_stitch.
* Revision 1.5 1997/12/20 21:29:09 avner
* separate use of 'bound' and 'cprof_bound'.
* Revision 1.4 1997/12/15 09:07:42 eval
* Change is_ximeric_edge to answer false if there is an RNA passing throue
* the edge.
* Revision 1.3 1997/12/01 11:20:47 eval
* Change the ximeric detection from looking at the edge width to looking at
* the number of supporting clones (i.e. ests from defferent clones)
* Revision 1.2 1997/11/30 08:05:06 ariels
* Added ChangeLog comment and static rcsid string.
* */
static char rcsid[] = "$Id: splice_graph.c,v 1.42 1998/07/01 13:44:32 eval Exp $";

#include <string.h>
#include <stdlib.h>

```

splice_graph.c

```

#include "olap_graph.h"
#include "splice_graph.h"
#include "custody_zones.h"
#include "est_table.h"
#include "error.h"
#include <assert.h>

extern FILE *variants_file;
extern char *transcripts[], p_graph_operations, p_seq_p_profile, contig_name[];
extern int bound_cutoff, indep_node_len, local_olap_mode;
extern char dirty_tails;
extern int cprof_bound;
extern int phase_no;
/* prototype of static functions */
static void node_free(splice_node *, int);
static void connect_nodes(splice_graph *graph, int source_node,
                          int target_node, int link);

static void splice_graph__mark_reachable(splice_graph *graph,
                                         int node, int target, int source, int *ma
rted);
/***** splice_node *****/
splice_node *node_new() /* create an empty node */
{
    splice_node *node;

    node = (splice_node *) malloc(sizeof(splice_node));
    if (node == NULL)
        err("node_new: memory failure trying to allocate for 'node'");
    node->seq = NULL;
    node->profile = NULL;
    node->out_degree = node->in_degree = 0;
    node->need_to_update = 0;
    node->start_update_into = node->start_update_with = 0;
    node->variant_depth = 0;
    node->align_str = NULL;
    node->align_list = NULL;
    return node;
}

void splice_node__copy(splice_node *src, splice_node **dup)
{
    int i;
    /*
     * if (((*dup)->seq == strdup(src->seq)) == NULL)
     *     err("splice_node__copy: memory failure trying to allocate for 'node->seq'")
     */
    *dup = node_new();
    (*dup)->id = src->id;
    (*dup)->profile = cprof_dup(src->profile);
    (*dup)->out_degree = src->out_degree;
    (*dup)->out_degrees = src->out_degrees;
    for (i = 0; i < src->out_degree; i++)
        (*dup)->out_neighbors[i] = src->out_neighbors[i];
}

int splice_graph__num_initial(splice_graph *graph)

```

splice_graph.c


```

(
    int i, num_init ;
    for (i = num_init = 0; i < graph->size; i++)
        if (splice_graph__is_initial_node(graph,i))
            num_init++;
    return num_init;
)

int splice_graph__num_terminal(splice_graph *graph)
(
    int i, num_termi ;
    for (i = num_termi = 0; i < graph->size; i++)
        if (splice_graph__is_ending_node(graph,i))
            num_termi++;
    return num_termi;
)

int splice_node__len(splice_node *node)
(
    return node->profile->len;
)

int splice_graph__out_nbr(splice_graph *graph, int node_idx, int nbr_idx)
(
    return graph->node_list[node_idx]->out_neighbors[nbr_idx].idx;
)

splice_node *node_init(cprof prof)
/* open a node with prof as it's profile, assign it's 'need_to_update' */
/* field to -1 for so that no updating will take place */
(
    splice_node *node = node_new();
    node->need_to_update=-1; /* since this operation is the equivalent to */
    node->profile = prof; /* updating (a 'null' profile)
    return node; /* added ariels */
)

void node_setid(splice_node *node, int nodeid) { /* set node id */
    node->id = nodeid;
}

static void node_free(splice_node *node, int free_profile_too)
(
    int i;
    void free_est_list(est_list);
    if (node->seq) free(node->seq);
    if (free_profile_too)
        cprof_free(node->profile);
    if (node->align_str) free(node->align_str);
    for (i=0; i<node->out_degree; i++)
        free_est_list(node->out_neighbors[i].ests_passing);
    free(node);
)

```

```

/***** splice_graph functions *****/
splice_graph *splice_graph_new() { /* create an empty graph */
    splice_graph *graph;

    graph = (splice_graph *) malloc(sizeof(splice_graph));
    graph->size = 0;
    return graph;
}

void splice_graph__copy(splice_graph *src, splice_graph **dup)
(
    int i;
    *dup = splice_graph_new();
    (*dup)->size = src->size;
    for (i=0; i<src->size; i++)
        splice_node__copy(src->node_list[i], &((*dup)->node_list[i]));
    void splice_graph_free(splice_graph *graph)
    (
        int i;
        if (graph==NULL)
            return;
        for (i = 0; i < graph->size; i++)
            node_free (graph->node_list[i],1);
        free (graph);
    )

    void splice_graph__align_lists_free(splice_graph *graph, int best_variant)
    (
        int i;
        for (i = 0; i < graph->size; i++)
        (
            if (i!=best_variant && graph->node_list[i]->align_list != NULL)
                align_data_list_free(graph->node_list[i]->align_list);
            graph->node_list[i]->align_list=NULL;
        )
    )

    int add_node(splice_graph *graph, splice_node *node)
    (
        if (graph->size>=MAX_NODES) {
            printf("#cluster %s :: can't add nodes to graph\n", contig_name);
            free(node);
            return -1;
        }
        graph->node_list[graph->size] = node;
        graph->node_list[graph->size]->id = graph->size;
        graph->size++;
        if (p_graph_operations == 'y' && phase_no == 2)
            p_graph_operations == 'v'
            printf("creating node %d\n", graph->size-1);
        return graph->size-1;
    )

    void add_edge(splice_graph *graph, int node1, int node2, int link)

```

```

    if (p_graph_operations == 'y' || p_graph_operations == 'v')
        printf("%d == (edge) ==> %d\n", node1, node2);
    if (node1 < 0 || node1 > graph->size || node2 < 0 || node2 > graph->size) {
        err("illegal call 'add_edge(graph,%d,%d)', node1, node2);
    }
    connect_nodes(graph, node1, node2, link);
}

void connect_nodes(splice_graph *graph, int source_node, int target_node,
                  int link)
{
    int nbr;
    arc *new_arc;

    for (nbr = 0; nbr < graph->node_list[source_node]->out_degree; nbr++)
        if (graph->node_list[source_node]->out_neighbors[nbr].idx == target_node)
            break;
    if (nbr < graph->node_list[source_node]->out_degree)
        graph->node_list[source_node]->out_neighbors[nbr].width += link;
    else {
        new_arc = &(graph->node_list[source_node]->
                    out_neighbors[graph->node_list[source_node]->out_degree]);
        new_arc->idx = target_node;
        new_arc->width = link;
        new_arc->is_ximeric = 0;
        new_arc->ests_passing = NULL;
        graph->node_list[source_node]->out_degree++;
        graph->node_list[target_node]->in_degree++;
    }
}

/*
 * is node1 --> node2 in graph?
 */
int splice_graph__is_directed_edge(splice_graph *graph, int node1, int node2)
{
    int i;

    if (node1 < 0 || node1 >= graph->size || node2 < 0 || node2 >= graph->size)
        err("splice_graph__is_directed_edge : internal error, call amit/avner");
    for (i = 0; i < graph->node_list[node1]->out_degree; i++)
        if (graph->node_list[node1]->out_neighbors[i].idx == node2)
            return 1;
    return 0;
}

/*
 * is there an edge (in any direction) between node1 and node2 ?
 */
int splice_graph__is_undirected_edge(splice_graph *graph,
                                     int node1, int node2)
{
    return splice_graph__is_directed_edge(graph, node1, node2) ||
        splice_graph__is_directed_edge(graph, node2, node1);
}

```

splice_graph.c

```

/*
 * Is there an undirected cycle?
 * Assume here graph is connected, and so a count of the edges gives us
 * the answer.
 */
int splice_graph__is_undirected_cycles(splice_graph *graph)
{
    int i, edge_no;

    for (i = edge_no = 0; i < graph->size; i++)
        edge_no += splice_graph__out_deg(graph, i);
    if (edge_no < graph->size - 1) return (-1);
    if (edge_no > graph->size - 1) return (1);
    else return (0);
}

int splice_graph__set_edge_and_ests_ximeric(splice_graph *graph,
                                             int node1, int node2)
{
    int i;
    est_list est;
    arc *edge;

    if (node1 < 0 || node1 >= graph->size || node2 < 0 || node2 >= graph->size) {
        err("splice_graph__is_directed_edge : internal error, call amit/avner");
    }
    for (i = 0; i < graph->node_list[node1]->out_degree; i++)
        if (graph->node_list[node1]->out_neighbors[i].idx == node2)
        {
            edge = &(graph->node_list[node1]->out_neighbors[i]);
            edge->is_ximeric = 1;

            for (est = edge->ests_passing; est != NULL; est = est->next)
                est_table__set_ximeric(est->est);

            return 1;
        }
    return 0;
}

int splice_graph__get_edge_width(splice_graph *graph, int node1, int node2)
{
    int i;

    if (node1 < 0 || node1 >= graph->size || node2 < 0 || node2 >= graph->size) {
        err("splice_graph__is_directed_edge : internal error, call amit/avner");
    }
    for (i = 0; i < graph->node_list[node1]->out_degree; i++)
        if (graph->node_list[node1]->out_neighbors[i].idx == node2)
            return graph->node_list[node1]->out_neighbors[i].width;
    return 0;
}

int splice_graph__in_deg(splice_graph *g, int node_id) {
    return g->node_list[node_id]->in_degree;
}

int splice_graph__out_deg(splice_graph *g, int node_id) {

```

splice_graph.c

```

    return g->node_list[node_id]->out_degree;
}

static est_list dup_est_list(est_list p)
{
    est_list head, q;
    head = NULL;
    while (p) {
        if (head)
            q = q->next = malloc(sizeof(struct est_list_s));
        else
            q = head = malloc(sizeof(struct est_list_s));
        if (q == NULL)
            err("memory failure in dup_est_list (%d bytes)",
                sizeof(struct est_list_s));
        q->est = p->est;
        q->next = NULL;
        p = p->next;
    }
    return head;
}

/* ===== */
int split_node(splice_graph *graph, int node_id, int offset, int *left, int *right)
{
    int i;
    splice_node *new_node;
    new_node = node_new();
    *left = node_id;
    if ((*right = add_node(graph, new_node)) == -1)
        return ret_err("calling function split_graph");
    if (p_graph_operations == 'y' || p_graph_operations == 'v')
        printf("split-node %d> %d---[%d]---->%d (continues upto %d)\n",
            node_id, *left, offset, *right, graph->node_list[node_id]->profile->len-1);
    graph->node_list[*right]->profile =
        cprof_split(graph->node_list[*left]->profile, offset);
    for (i=0; i<graph->node_list[*left]->out_degree; i++)
    {
        graph->node_list[*right]->out_neighbors[i].idx =
            graph->node_list[*left]->out_neighbors[i].idx;
        graph->node_list[*right]->out_neighbors[i].width =
            graph->node_list[*left]->out_neighbors[i].width;
        graph->node_list[*right]->out_neighbors[i].is_ximetric =
            graph->node_list[*left]->out_neighbors[i].is_ximetric;
        graph->node_list[*right]->out_neighbors[i].ests_passing =
            graph->node_list[*left]->out_neighbors[i].ests_passing;
    }
    graph->node_list[*right]->out_degree = graph->node_list[*left]->out_degree;
    graph->node_list[*right]->in_degree = 1;
    graph->node_list[*left]->out_degree = 1;
    graph->node_list[*left]->out_neighbors[0].idx = *right;
}

```

```

graph->node_list[*left]->out_neighbors[0].width =
    graph->node_list[*right]->profile->link[0];
graph->node_list[*left]->out_neighbors[0].is_ximetric = 0;
graph->node_list[*left]->out_neighbors[0].ests_passing = NULL;
return 1;
}

#ifdef COMPACTIZATION
int compare_floats(float a, float b);
int compare_floats(float a, float b)
{
    return a-b;
}

compactize_graph(splice_graph *graph)
{
    float length_node_array[MAX_NODES];
    int node;
    for (node=0; node<graph->size; node++)
        length_node_array[node] = 1./(node+1) + splice_node__len(graph->node_list(node));
    qsort(length_node_array, graph->size, sizeof(float), compare_floats);
    /*
    for (idx=0; idx<graph->size; idx++)
    {
        if (splice_graph__out_degree(graph, node) == 1 && splice_node__len(graph->node_list(node)) < TRASH_NODE)
            splice_graph__delete_node(graph, node, 1);
        if (splice_graph__out_degree(graph, node) == 1 && splice_node__len(graph->node_list(node)) < TRASH_NODE)
            /*
        */
        #endif
    }
    /*
    * unify_needed_nodes - unify nodes 1 and 2 if you cannot enter 2 not from 1
    * and you cannot exit 1 not toward 2
    * i.e. out_degree(1) == in_degree(2) == 1 and 1 is not an ending node
    * and 2 is not an initial node.
    */
    void unify_needed_nodes(splice_graph *graph)
    {
        int node, nbr;
        find_nodes_types(graph);
        for (node=0; node<graph->size; node++)
            while (graph->node_list[node]->out_degree==1 &&
                splice_graph__in_deg(graph,
                    nbr = splice_graph__out_nbr(graph, node, 0)) == 1 &&
                    !splice_graph__is_ending_node(graph, node) &&
                    !splice_graph__is_initial_node(graph, nbr))
                (void)unify_nodes(graph, node, nbr);
    }
}

```

A-167

```

int unify_nodes(splice_graph *graph, int node_left, int node_right)
/*
 * given two nodes of index node_left and node_right, with in-degree/out-degree
 * = 1, respectively, unify them to one node, indexed node_left, whose
 * in-neighbors are those of the old node_left, and his out-neighbors are
 * node_right's.
 * To complete the operation the right node is deleted
 * Fix: Due to the deletion the index of the left node could be changed,
 * (if he is the largest index) so we return this index if it changed and -1
 * if not.
 */

```

```

    int i;
    void free_est_list(est_list);

    if (p_graph_operations == 'v' || p_graph_operations == 'v')
        printf("unify nodes %d-->%d\n", node_left, node_right);
    cprof_glue(graph->node_list[node_left]->profile,
               graph->node_list[node_right]->profile, e_after,
               graph->node_list[node_left]->out_neighbors[0].width);

/* ariels bugfix 26/11/97 */
for(i=0; i<graph->node_list[node_left]->out_degree; i++)
    free_est_list(graph->node_list[node_left]->out_neighbors[i].ests_passing);

for(i=0; i<graph->node_list[node_right]->out_degree; i++)
{
    graph->node_list[node_left]->out_neighbors[i].idx =
        graph->node_list[node_right]->out_neighbors[i].idx;
    graph->node_list[node_left]->out_neighbors[i].width =
        graph->node_list[node_right]->out_neighbors[i].width;

/* ariels bugfix 26/11/97 */
    graph->node_list[node_left]->out_neighbors[i].ests_passing =
        graph->node_list[node_right]->out_neighbors[i].ests_passing;
    graph->node_list[node_right]->out_neighbors[i].ests_passing = NULL;
    graph->node_list[node_left]->out_neighbors[i].is_ximeric =
        graph->node_list[node_right]->out_neighbors[i].is_ximeric;
}

graph->node_list[node_left]->out_degree = graph->node_list[node_right]->out_de
gree;

return graph->delete_node(graph, node_right, 0); /* 0 because we don't
/* want to free the
/* profile of the node
*/
}

/* Return the index of the node that was changed due to this deletion,
 * -1 if there non such (i.e. node_idx == graph->size)
 */
int graph->delete_node(splice_graph *graph, int node_idx, int profile_mem_free)
{
    int replacing_node = -1;

    node_free(graph->node_list[node_idx], profile_mem_free);
    if (node_idx < --graph->size) {
        graph->node_list[node_idx] = graph->node_list

```

splice_graph.c

```

[replacing_node = graph->size];
graph->node_list[node_idx]->id = node_idx;
/* delete occurrences of node_idx in neighboring-lists */
replace_occurrences_in_neighbors_list(graph, node_idx, -1);
/* replace occurrences of replacing_node in node_idx, in neighboring-lists */
replace_occurrences_in_neighbors_list(graph, replacing_node, node_idx);
}
else
/* delete occurrences of node_idx in neighboring-lists */
replace_occurrences_in_neighbors_list(graph, node_idx, -1);
return replacing_node;
}

```

```

void replace_occurrences_in_neighbors_list(splice_graph *graph, int old_idx, int ne
w_idx)
/*
 * replace all occurrences of old_idx in all neighbors-lists with new_idx
 * if new_idx==1 then we think of it as a delete operation
 */
{
    int node_nbr;

    for (node=0; node<graph->size; node++)
        for (nbr=0; nbr < graph->node_list[node]->out_degree; nbr++)
            if (graph->node_list[node]->out_neighbors[nbr].idx==old_idx)
                if (new_idx>=0)
                    graph->node_list[node]->out_neighbors[nbr].idx=new_idx;
                else
                    graph->node_list[node]->out_neighbors[nbr].idx =
                        graph->node_list[node]->out_neighbors
                        [graph->node_list[node]->out_degree].idx;
}

/* for an edge to be considered ximeric, it should be of width 1 (exactly
 * one est should support it), to be a bridge (in the undirected sense)
 * and not to be the only edge out of it's source,
 * and not to be the only edge into of it's target
 */
int splice_graph__is_ximeric_edge(splice_graph *graph, int source_node,
int target_node)
{
    int node_nbr, size, target_component, *marked;
    if ((marked = (int *)calloc(graph->size, sizeof (int))) == NULL)
        err("splice_graph__is_ximeric_edge:
        "memory failure trying to allocate for 'marked'");

    if (splice_graph__is_rna_supported(graph, source_node, target_node) ||
        !splice_graph__is_one_clone(graph, source_node, target_node) ||
        splice_graph__out_deg(graph, source_node)<2 ||
        splice_graph__in_deg(graph, target_node)<2) {
        free(marked);
        return 0;
    }
    splice_graph__mark_reachable(graph, target_node,
                                target_node, source_node, marked);
}

```

splice_graph.c

```

for (node=0; size_target_component==0; node<graph->size; node++)
    if (marked[node]) size_target_component++;
free(marked);
return (size_target_component>0 && size_target_component<graph->size);
}

/*EP*/
int splice_graph__is_one_clone(splice_graph *graph,
    int source_node,
    int target_node)
{
    int i;
    est_list est1, est2;
    arc *edge = NULL;
    char *clone1, *clone2;

    if (source_node<0 || source_node>graph->size ||
        target_node<0 || target_node>graph->size) {
        err("is_one_clone : internal error, call amit/avner");
    }
    for(i=0; i<graph->node_list[source_node]->out_degree; i++)
        if (graph->node_list[source_node]->out_neighbors[i].idx == target_node)
            edge = &(graph->node_list[source_node]->out_neighbors[i]);
    if (edge == NULL) return 0;
    if (edge->ests_passing == NULL) return 0;
    for(est1 = edge->ests_passing; est1->next != NULL; est1 = est1->next)
    {
        clone1 = est_table__get_clone( est1->est );
        for(est2 = est1->next; est2 != NULL; est2 = est2->next)
        {
            clone2 = est_table__get_clone( est2->est );
            if (clone1 == NULL || clone2 == NULL ||
                strcmp(clone1, clone2))
                return 0;
        }
    }
    return 1;
}

/*
 * Return true iff there is an RNA in the ests_passing list
 */
int splice_graph__is_rna_supported(splice_graph *graph,
    int source_node,
    int target_node)
{
    int i;
    est_list est;
    arc *edge = NULL;

    if (source_node<0 || source_node>graph->size ||
        target_node<0 || target_node>graph->size) {
        err("is_rna_supported : internal error, call amit/avner");
    }
    for(i=0; i<graph->node_list[source_node]->out_degree; i++)
        if (graph->node_list[source_node]->out_neighbors[i].idx == target_node)
            edge = &(graph->node_list[source_node]->out_neighbors[i]);
    if (edge == NULL) return 0;
    if (edge->ests_passing == NULL) return 0;
    for(est = edge->ests_passing; est != NULL; est = est->next)

```

```

    if (est_table__is_rna(est->est))
        return 1;
    return 0;
}

/*
 * mark all nodes reachable from node, in the (undirected) graph
 * WITHOUT using the (undirected) edge <source> <--> <target>.
 */
static void splice_graph__mark_reachable(splice_graph *graph,
    int node, int target, int source,
    int *marked)
{
    int nbr;
    marked[node]=1;
    for (nbr=0; nbr<graph->size; nbr++)
        if ( !marked[nbr] &&
            splice_graph__is_undirected_edge(graph, node, nbr) &&
            ! (node == source && nbr == target || node == target && nbr == source) )
            /* don't use the (undirected) edge source<-->target */
            splice_graph__mark_reachable(graph, nbr, target, source, marked);
}

/*
 * is the graph connected in the undirected sense.
 */
int splice_graph__is_connected(splice_graph *graph)
{
    int i, *marked;
    if ((marked = (int *)calloc(graph->size, sizeof (int))) == NULL)
        err("splice_graph__is_connected: "
            "memory failure trying to allocate for 'marked'");
    splice_graph__mark_reachable(graph, 0, -1, -1, marked);
    for (i=0; i<graph->size; i++)
        if (!marked[i]) {
            free(marked);
            return 0;
        }
    free(marked);
    return 1;
}

/*
 * find all the graph connective components, COMPONENTS is
 * an array of graph->size ints, the function set the i'th entry with
 * the components index of the node i.
 * return the number of components.
 */
int splice_graph__find_connective_components(splice_graph *graph,
    int *components)
{
    int i, j, comp_idx=0, *marked;
    if ((marked = (int *)malloc(sizeof (int) * graph->size)) == NULL)
        err("splice_graph__is_connected: "
            "memory failure trying to allocate for 'marked'",
            sizeof(int) * graph->size);

```

```

for (i=0; i<graph->size; i++)
    components[i]=-1;
for (i=0; i<graph->size; i++) {
    if (components[i] == -1) {
        for (j=0; j<graph->size; j++)
            marked[j]=0;
        splice_graph_mark_reachable(graph, i, -1, -1, marked);
        for (j=0; j<graph->size; j++)
            if (marked[j]) {
                if (components[j] != -1)
                    err("Internal err in find_connective_components\n");
                components[j] = comp_idx;
            }
        comp_idx++;
    }
}
free(marked);
return comp_idx;
}

```

```

/*****
void update_node(splice_node *node, cprof entering_profile)
{
    if (p_profile == 'Y')
        printf("update node %d, offset = %d, align_string = %s\n", node->id,
            node->start_update_into, node->align_str);
    cprof_update(node->profile, entering_profile,
        node->start_update_into, node->start_update_with,
        node->align_str, 0);
    free(node->align_str);
    node->align_str = NULL;
    node->need_to_update = 0;
}

```

```

void fix_update_node_parameters(splice_node *node,
    int st_offset_into, int st_offset_with,
    char *str)
{
    if (node->need_to_update==0) /* otherwise this node was updated from 'the' */
        /* other side' and updating again will be */
        /* an even wrong (due to an extension earlier) */
        /* that make offset no longer true) operation */
    {
        node->need_to_update=1;
        node->start_update_into = st_offset_into;
        node->start_update_with = st_offset_with;
        node->align_str = strdup(str);
    }
}

```

```

void extend_node(splice_graph *graph, int node_id, cprof entering_profile,
    int start, int end, int offset, int direction)
{
    cprof new_prof;
    char c;
    new_prof = cprof_segment(entering_profile, start, end);
}

```

splice_graph.c

```

if (direction==LEFT_EXTENSION)
{
    cprof_cut_head(graph->node_list[node_id]->profile, offset);
    cprof_glue(graph->node_list[node_id]->profile, new_prof, e_before,
        new_prof->link[new_prof->len]);
}
else
{
    cprof_cut_tail(graph->node_list[node_id]->profile, offset);
    cprof_glue(graph->node_list[node_id]->profile, new_prof, e_after,
        new_prof->link[0]);
}
}

/*****
align_data *align_node_to_cprof (splice_node *node, cprof entering_profile)
{
    cprof_align cp_align, cp_align_list;
    char *tmp_seq, *align_string;
    int i;
    align_data *adl, *ad;

    cp_align_list = cprof_alignment(node->profile, entering_profile,
        SEPARATE_HILLTOPS|local_olap_mode);
    adl = cprof_align_to_align_data(cp_align_list, node->profile, entering_profile,
        node->id);
    cprof_align_free(cp_align_list);
    return adl;
}

align_data *cp_prof_align_to_align_data(cprof_align cpal, cprof pl, cprof p2,
    int node_id)
{
    cprof_align cp_align;
    double id_percent, sim_percent;
    align_data *ad, *adl=NULL, *last;
    for (cp_align = cpal; cp_align != NULL; cp_align = cp_align->next) {
        /* process alignment */
        if (cp_align->score < cprof_bound) continue;
        cprof_calc_id_sim(pl, p2, cp_align->start1, cp_align->start2,
            cp_align->str, &id_percent, &sim_percent);
        ad = align_data_new (NULL, -1, -1);
        ad->node_id = node_id;
        ad->score = cp_align->score;
        ad->start_node = cp_align->start1;
        ad->start_est = cp_align->start2;
        ad->end_node = cp_align->end1;
        ad->end_est = cp_align->end2;
        ad->prefix = (pure_seglen(pl, 0, cp_align->start1 - 1) <
            indep_node_len);
        ad->suffix = (pure_seglen(pl, cp_align->end1 + 1, pl->len-1)
            < indep_node_len);
        ad->align_str = strdup(cp_align->str);
        ad->id_percent = id_percent;
    }
}

```

splice_graph.c

```

ad->sim_percent = sim_percent;
if(adl == NULL)
    adl = last = ad;
else {
    last->next = ad;
    last = ad;
}
return adl;
}

void graph_print(splice_graph *graph)
{
    int i;
    for (i = 0; i < graph->size; i++)
        splice_node_print(graph->node_list[i], i);
}

void splice_node_print(splice_node *node, int id)
{
    int nbr, est_idx, dummy, *ests;
    char *tmp;

    if (p_graph_operations == 'y' || p_graph_operations == 'v') {
        if (node->variant_depth > 0) {
            /*
            ests = (int*)malloc(node->variant_depth * sizeof(int));
            assert(cprof_ids(node->profile, ests, node->variant_depth) ==
                    node->variant_depth);
            */

            ests = (int*)malloc(node->profile->width * sizeof(int));
            cprof_ids(node->profile, ests, node->profile->width);

            printf(">variant%d #LN %d #DPH %d |\n",
                    id, node->profile->len, node->variant_depth);
            for (est_idx = 0; est_idx < node->variant_depth; est_idx++)
                printf(" %d", ests[est_idx]);
            printf(" |\n");
            free(ests);
        }
        if (variants_file) {
            fprintf(variants_file, ">variant%d CU %s #CN %s #LN %d\n",
                    id, config_name, node->profile->len, node->variant_depth,
                    write_sequence(variants_file, tmp,
                                   cprof_compute_assembly(node->profile, 0, &dummy, &dummy)));
            free(tmp);
        }
    }
    else
        printf(" node %d::\nlen %d pure-len %d\n",
                id, splice_node->len(node),
                pure_seglen(node->profile, 0, node->profile->len - 1));

    if (p_seq == 'y') {
        write_sequence(stdout, tmp, cprof_compute_assembly
                      (node->profile, 0, &dummy, &dummy));
        free(tmp);
    }
}

```

```

if (p_profile == 'y')
    cprof_print(node->profile);
}
if (p_graph_operations == 'y' || p_graph_operations == 'v') {
    for (nbr=0; nbr < node->out_degree;nbr++) {
        printf("%s",nbr?",";"edges to nodes ");
        printf("%d(%d)",node->out_neighbors[nbr].idx,
            node->out_neighbors[nbr].width);
    }
    printf("\n");
}
)
)

void graph__print_cprofs(splice_graph *graph, FILE *fp)
{
    int i;

    for (i = 0; i<graph->size; i++)
        splice_node__print_cprof(graph->node_list[i],i,fp);
}

void splice_node__print_cprof(splice_node *node, int id, FILE *fp)
{
    printf("      node %d:\nlen %d\n" id.splice_node__len(node));
    cprof_dump(node->profile);
}

int check_czm_for_order_violation(algn_data *czm,int graph_size)
{
    int i, last_algn[MAX_NODES];

    for(i=0;i<graph_size;last_algn[i++]= 0);

    while (czm) {
        if (czm->node_id!=FREE_SEG)
            extern int algn_intersec_high;
        if (czm->start_node < last_algn[czm->node_id] - algn_intersec_high)
            return 0;
        else
            last_algn[czm->node_id] = czm->end_node+1;
    }
    czm = czm->next;
}

return 1;
}

/*
** excellent_overlap ::
**   the alignment is good enough to know we want to send the current
**   est to the current node's variant.
*/
int excellent_overlap(splice_node *node,int est_idx)
{
    int tmp;

```



```

return ((tmp = overlap_score(node, est_idx)) != -1 &&
        tmp < indep_node_len / 2);
)

/*****
*/
/* overlap_score ::
 * if we have an overlap, give the score to estimate how good it is. namely,
 * the length of the longer overhang-tail. Otherwise return -1
 */
int overlap_score(splice_node *node, int est_idx)
{
    if (node->align_list == NULL || node->align_list->next != NULL)
        /* to be an overlap, there should be exactly one alignment element */
        return -1;
    return
        max(
            min(
                node->align_list->start_est,
                pure_seglen(node->profile, 0, node->align_list->start_node - 1)
            ),
            min(
                est_table__len(est_idx) - 1 - node->align_list->end_est,
                pure_seglen(node->profile, node->align_list->end_node + 1,
                    splice_node__len(node) - 1)
            )
        );
}

/* 'make' est <est_idx> overlap with variant node <node_id> :
 * do nothing if they (fuzzily) overlap as they are.
 * otherwise a dirty-tail must be cut in either the est or the node. If
 * both are a possibility, cut the one that require minimal cutting for
 * turning the alignment into an overlap.
 */
int cut_dirty_overhangs(splice_graph *graph, int node_id,
    int est_cprof, int est_idx)
{
    int est_head_ohang, est_tail_ohang, est_clean_head,
    est_clean_tail, est_cut_tail_len,
    node_head_ohang, node_tail_ohang, node_clean_head,
    node_clean_tail, node_cut_tail_len,
    est_head_cupoint=-1, node_head_cupoint=-1,
    est_tail_cupoint=-1, node_tail_cupoint=-1, rc=0;
    splice_node *node;
    cprof prof;
    int node_est_idx;

    node = graph->node_list[node_id];
    prof = node->profile;

    /* head-side (5') */
    node_head_ohang = node->align_list->start_est;
    node_head_cupoint = pure_seglen(prof, 0,
        node->align_list->start_node - 1);
    if (min(est_head_ohang, node_head_ohang) >= indep_node_len) {

```

splice_graph.c

```

/* the next 2 variables should be read as "where did the alignment start
from in the clean version of the est/node" (can be negative) */
est_clean_head = node->align_list->start_est -
    est_table__first_clean(est_idx);
node_clean_head = pure_seglen(prof,
    node__first_clean(prof),
    node->align_list->start_node - 1);

/* which of the cuts makes an overlap, and where should the cut be */
if (est_clean_head < indep_node_len)
    est_head_cupoint = min(est_head_ohang, est_table__first_clean(est_idx));
if (node_clean_head < indep_node_len)
    node_head_cupoint = min(node_head_ohang,
        cprof__first_clean(prof));

/* which cut is the best ? */
if (est_head_cupoint != -1 && (node_head_cupoint == -1 ||
    est_head_cupoint < node_head_cupoint)) {
    cprof_cut_head(est_cprof, est_head_cupoint);
    est_table__cut_head(est_idx, est_head_cupoint);
    cprof_fix_id_after_cut(est_cprof, est_idx, est_head_cupoint);
    node->align_list->start_est = est_head_cupoint;
    node->align_list->end_est = est_head_cupoint;
    printf("CUTTING head [%d] of est %d\n", est_head_cupoint, est_idx);
}
else {
    assert(node_head_cupoint != -1);
    /* We got the id of the dirty est in the node tail and cut it.
     * cprof_pos_ids return the number of ids in pos=0 - it must be 1 */
    assert(cprof_pos_ids(prof, 0, &node_est_idx, 1) == 1);
    cprof_cut_head(prof, node_head_cupoint);
    cut_est_head(node_est_idx, node_head_cupoint, graph);
    node->align_list->start_node = node_head_cupoint;
    node->align_list->end_node = node_head_cupoint;
    printf("CUTTING head [%d] of variant %d\n", node_head_cupoint, node_id);
}
rc=1;
}

/* tail side (3') */
est_tail_ohang = est_table__len(est_idx) - 1 - node->align_list->end_est;
node_tail_ohang = pure_seglen(prof, node->align_list->end_node + 1,
    prof->len-1);
if (min(est_tail_ohang, node_tail_ohang) >= indep_node_len) {
    est_clean_tail = est_tail_ohang - (est_table__first_dirty(est_idx) -
        est_table__len(est_idx));
    node_clean_tail = node_tail_ohang - pure_seglen(prof,
        node__last_clean(prof) + 1,
        prof->len-1);
    /* which of the cuts make an overlap, and where should the cut be */
    if (est_clean_tail < indep_node_len) {
        est_tail_cupoint = max(node->align_list->end_est + 1,
            est_table__first_dirty(est_idx));
        est_cut_tail_len = est_table__len(est_idx) - est_tail_cupoint;
    }
    if (node_clean_tail < indep_node_len) {
        node_tail_cupoint = max(node->align_list->end_node + 1,
            cprof__last_clean(prof));
        node_cut_tail_len = pure_seglen(prof, node_tail_cupoint, prof->len-1);
    }
    if (est_tail_cupoint != -1 && (node_tail_cupoint == -1 ||
        est_cut_tail_len > node_cut_tail_len)) {

```

splice_graph.c

```

/* We cut the est */
cprof_cut_tail(est_cprof, est_cut_tail_len);
cut_est_tail(est_idx, est_cut_tail_len, graph);
printf("CUTTING tail [%d] of est %d\n", est_tail_outpoint, est_idx);
}
else { /* We cut the node */
/* We get the id of the dirty est in the node tail and cut it.
* cprof_pos_ids return the number of ids in the last position -
* it must be 1 */
assert(cprof_pos_ids(prof, cprof->len-1, &node_est_idx, 1) == 1);
cprof_cut_tail(prof, node_cut_tail_len);
est_table__cut_tail(node_est_idx, node_cut_tail_len);
printf("CUTTING tail [%d] of variant %d\n", node_tail_outpoint, node_id);
}
rc=1;
}
node->align_list->prefix = node->align_list->suffix = 1;
return rc;
}

int clean_overlap_score(splice_node *node, int est_idx)
{
    if (node->align_list==NULL || node->align_list->next==NULL)
/* to be an overlap, there should be exactly one alignment element */
/* need to improve: the right thing here would be to see if by */
/* cancelling the alignments that are in a dirty part, we so get */
/* one alignment element */
return -1;
}
max(
    min(
        node->align_list->start_est_est_table__first_clean(est_idx),
        pure_seglen(node->profile,
            cprof_first_clean(node->profile),
            node->align_list->start_node - 1)
        ),
        min(
            est_table__first_dirty(est_idx)-1-node->align_list->end_est,
            pure_seglen(node->profile, node->align_list->end_node + 1,
                cprof_last_clean(node->profile))
            )
        );
}
/*
*/
int get_best_clean_variant(splice_graph *graph, int est_idx)
{
    int i, best_idx = -1, best_score = -1, tmp;

    for (i=0; i< graph->size; i++)
        if ((tmp = clean_overlap_score(graph->node_list[i], est_idx)) != -1
            && (best_score == -1 || tmp < best_score))
        {
            best_idx = i;
            best_score = tmp;
        }
    if (best_score < indep_node_len)
        return best_idx;
}

```

```

else
    return -1;
}
/*
*/
int get_best_dirty_variant(splice_graph *graph, int est_idx)
{
    int i, best_idx = -1, best_score = -1, tmp;

    for (i=0; i< graph->size; i++)
        if ((tmp = overlap_score(graph->node_list[i], est_idx)) != -1
            && (best_score == -1 || tmp < best_score))
        {
            best_idx = i;
            best_score = tmp;
        }
    if (best_score < indep_node_len)
        return best_idx;
    else {
        printf("best not taken: %d\n", best_idx);
        printf("est clean part: %d-%d (full len %d)\n",
            est_table__first_clean(est_idx),
            est_table__first_dirty(est_idx)-1,
            est_table__len(est_idx));
        printf("variant %d clean part: %d-%d (full len %d)\n", best_idx,
            cprof_first_clean(graph->node_list[best_idx]->profile),
            cprof_last_clean(graph->node_list[best_idx]->profile),
            splice_node__len(graph->node_list[best_idx]));
        align_data_list__print(graph->node_list[best_idx]->align_list);
        return -1;
    }
}
/*
* of all alignments, return the index of the node which aligns in an
* overlapping way, and with minimal overhang-tail, or -1 if no such
* node exists
*/
int get_best_variant(splice_graph *graph, int est_idx)
{
    int best_idx;

    best_idx = get_best_dirty_variant(graph, est_idx);
    if (best_idx == -1 && dirty_tails == 'y')
        best_idx = get_best_clean_variant(graph, est_idx);
    return best_idx;
}

void splice_graph__increment_variant_depth(splice_graph *graph, int idx)
{
    graph->node_list[idx]->variant_depth++;
}

char *get_alignment_string(hilltop *ht, char *second_seq)
{
    int i;
    char *string;
}

```

```

if ((string = (char *) malloc(ht->len + 1 + strlen(second_seq))) == NULL)
    err("get_alignment_string: memory failure trying to allocate for 'string'");
for(i = 0; i < ht->len; i++) {
    if (ht->x[i] != '-')
        string[i] = ht->y[i];
    else
        string[i] = ht->y[i] - 'A' + 'a';
}
string[ht->len] = 0;
return string;
}

/* Return matrix representation of graph
*/
int *splice_graph__get_matrix(splice_graph *graph)
{
    int *res;
    int i;
    int j;

    if ((res = (int *) calloc(graph->size*graph->size, sizeof(int))) == NULL)
        return NULL;

    for(i=0; i<graph->size; i++)
        for(j=0; j < graph->node_list[i]->out_degree; j++)
            res[i*graph->size + graph->node_list[i]->out_neighbors[j].idx] = 1;

    return res;
}

/* free an est_list (despite the name, it's part of a splice-graph node)
*/
void free_est_list(est_list p)
{
    est_list q;

    while(p) {
        q = p->next;
        free(p);
        p = q;
    }

    /* add est# to list of ests passing between nodes
    */
    void splice_graph__add_est_between_nodes(splice_graph *graph,
        int est_no,
        int from, int to)
    {
        int nbr;
        est_list node;

```

splice_graph.c

```

/* Find index of TO as FROM's neighbour */
for(nbr=0; nbr<graph->node_list[from]->out_degree; nbr++)
    if (graph->node_list[from]->out_neighbors[nbr].idx == to)
        break;
if (nbr == graph->node_list[from]->out_degree)
    err("Internal error in splice_graph__add_est_between_nodes: "
        "not neighbours!");
else if ((node = malloc(sizeof(struct est_list_s))) == NULL)
    err("memory failure in splice_graph__add_est_between_nodes for %d bytes",
        sizeof(struct est_list_s));

node->est = est_no;
node->next = graph->node_list[from]->out_neighbors[nbr].ests_passing;
graph->node_list[from]->out_neighbors[nbr].ests_passing = node;
}

/*=====
*/
/* connect_stitch:
* point1 is the ending position of some sequence <left> in the graph,
* and point2 is the starting position of some other sequence <right> in the
* graph. Here we use the fact that <left> if to the left of <right> in the
* sequence we update the graph by. Therefore, there should be a connection
* between those positions. Splittings and unification might be needed here
* to express this connection in a loyal way.
* Note there are special points whose position must be preserved through
* whatever operations we do here.
*/

int connect_stitch(splice_graph *graph,
    graph_point *point1, graph_point *point2, int link,
    graph_point **points_to_update,
    char *indentation)
{
    int left,right,node_id,replaced_node;
    unsigned int left_len;

    if(point1 == NULL || point2 == NULL) return 0;

    right = point2->node;
    left = point1->node;
    delete_point(points_to_update,point1);
    delete_point(points_to_update,point2);
    add_edge(graph,left,right,link);

    /*
    if(graph->node_list[left]->out_degree==1 &&
    graph->node_list[right]->in_degree==1 &&
    left != right)
    {
        left_len = splice_node__len(graph->node_list[left]);
        replaced_node = unify_nodes(graph,left,right);
        update_points_list_after_unify(points_to_update,left,right,
            left_len,replaced_node);
    }
    */
    return 1;
}

```

splice_graph.c

```

    )

void align_data_list_identity_filter(align_data **list, int bound) {
    align_data *ad, *prev=NULL, *delete_list=NULL;

    for(ad=*list; ad != NULL; ) {
        if(ad->id_percent < bound) { /* delete ad and remove it from the list */
            printf("overrule alignment: <ad, id>,,&id,n8s\n(%.2f)\n",
                ad->start_node, ad->end_node, ad->start_est, ad->end_est,
                ad->align_str, ad->id_percent);

            if (prev == NULL) {
                *list = (*list)->next;
                ad->next=NULL;
                align_data_list_free(ad);
                ad=*list;
            } else {
                prev->next = ad->next;
                ad->next=NULL;
                align_data_list_free(ad);
                ad=prev->next;
            }
            else {
                prev=ad;
                ad=ad->next;
            }
        }
    }

    int splice_graph__is_initial_node(splice_graph *graph, int node)
    {
        return graph->node_list[node]->type & INITIAL;
    }

    int splice_graph__is_ending_node(splice_graph *graph, int node)
    {
        return graph->node_list[node]->type & ENDING;
    }

    int find_nodes_types(splice_graph *graph)
    {
        int *order_rel;
        int i, j, N = graph->size;
        splice_node *node;

        if((order_rel = (int*)malloc(N*N*sizeof(int))) == NULL)
            err("memory failer in find_initial_nodes");

        for(i=0; i<N; i++) {
            graph->node_list[i]->type = NON;
            for(j=0; j<N; j++)
                order_rel[i*N+j] = SMALLER;
        }

        for(i=0; i<graph->size; i++) {

```

```

    node = graph->node_list[i];
    for(j=0; j<graph->size; j++)
        graph->node_list[j]->marked = 0;

    find_all_sons(i, node, graph, order_rel);

    for(i=0; i<N; i++)
        for(j=0; j<N; j++)
            if(order_rel[i*N+j] == BIGGER && order_rel[j*N+i] == BIGGER)
                order_rel[i*N+j] = order_rel[j*N+i] = EQUAL;

    /* Mark all the initial nodes */
    for(j=0; j<N; j++) {
        node = graph->node_list[j];
        node->type |= INITIAL;
        for(i=0; i<N; i++) {
            if(order_rel[i*N+j] == BIGGER) {
                node->type &= -INITIAL;
                break;
            }
        }
    }

    /* Mark all the ending nodes */
    for(i=0; i<N; i++) {
        node = graph->node_list[i];
        node->type |= ENDING;
        for(j=0; j<N; j++) {
            if(order_rel[i*N+j] == BIGGER) {
                node->type &= -ENDING;
                break;
            }
        }
    }

    free(order_rel);
    return 1;
}

int find_all_sons(int root_node, splice_node *node, splice_graph *graph, int *order_rel)
{
    int nbr;

    if (!node->marked) {
        node->marked = 1;
        order_rel[root_node*graph->size + node->id] = BIGGER;
        for(nbr=0; nbr<node->out_degree; nbr++)
            find_all_sons(root_node, graph->node_list[node->out_neighbors[nbr].idx],
                graph, order_rel);
    }
    return 1;
}

void cut_est_head(int est_idx, int est_head_cutpoint, splice_graph *graph)
{
    int i;
    est_table__cut_head(est_idx, est_head_cutpoint);
    for(i=0; i<graph->size; i++) {
        cprof_fix_id_after_cut(graph->node_list[i]->profile, est_idx,
            est_head_cutpoint);
    }

```

```
    }  
}  
void cut_est_tail(int est_idx, int est_tail_len, splice_graph *graph)  
{  
    est_table__cut_tail(est_idx, est_tail_len);  
}
```

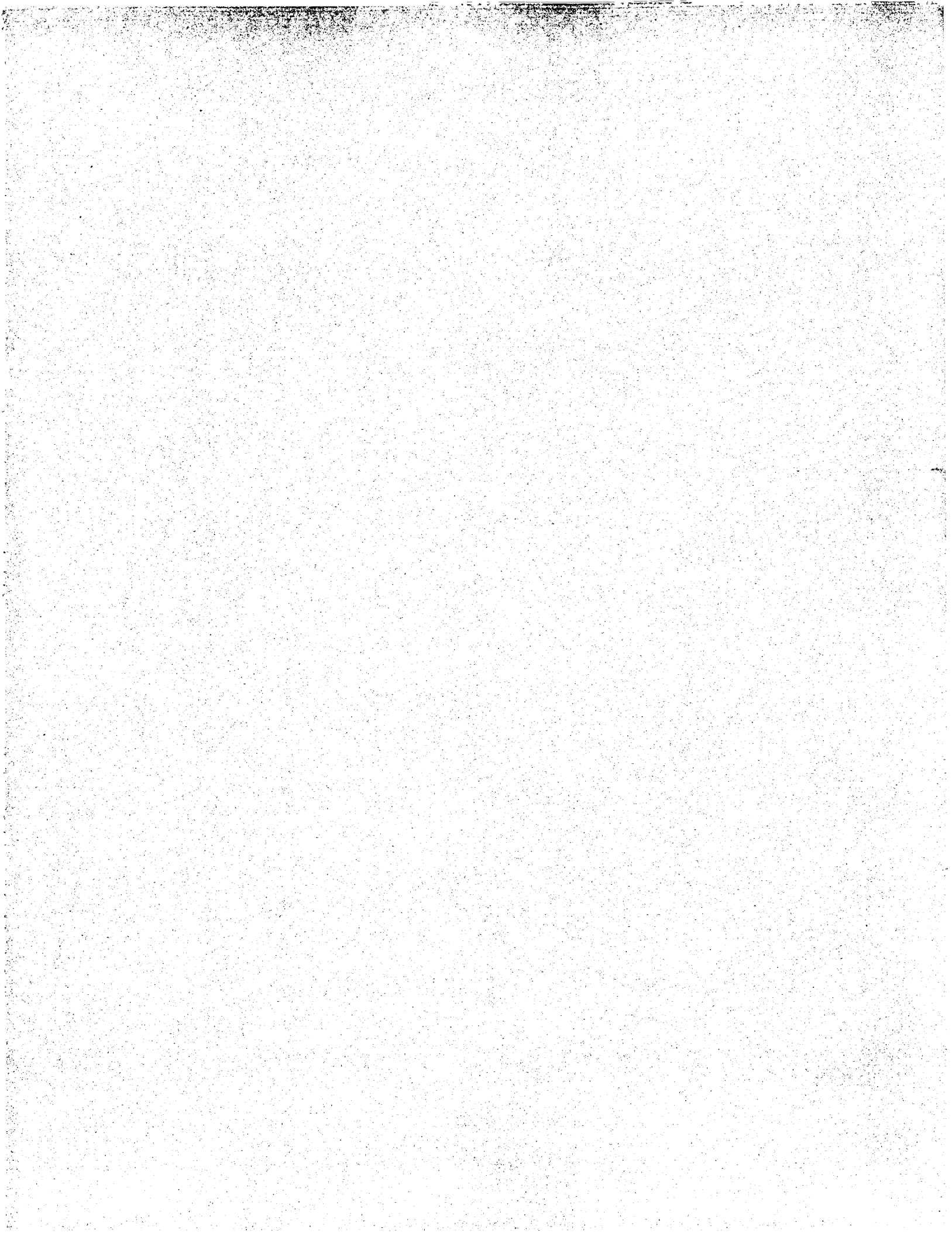
```

/*
 * $Id: align_data.h,v 1.4 1998/04/29 10:34:40 eyal Exp $
 * $Log: align_data.h,v $
 * Revision 1.4 1998/04/29 10:34:40 eyal
 * adding field 'marked_cancelled' to struct align_data.
 *
 * Revision 1.3 1998/01/11 10:58:57 eyal
 * Adding fields id_percent and sim_percent
 *
 * Revision 1.2 1997/11/30 07:39:18 ariels
 * Added Id and ChangeLog comments.
 *
 */

#ifndef ALIGN_DATA
#define ALIGN_DATA
typedef struct align_data {
    struct align_data *next;
    int node_id;
    int score;
    int start_node,end_node; /* of alignments in the node's side */
    int start_est,end_est; /* of alignments in the est's side */
    int prefix,suffix; /* is the left/right of node in alignment? */
    int marked_cancelled; /* to be used when scanning the list and deciding */
    double id_percent; /* percentage of identity in the alignment */
    double sim_percent; /* percentage of similarity in the alignment */
    char *align_str; /* both strands of alignment into one string which */
    struct align_data *align_data;
} align_data;

#endif

```




```

/*
 * align_prm.h -- Probabilistic parameters for cprof_align
 *
 * ariels 27/10/97
 */

/*
 * $Id: align_prm.h,v 1.9 1998/06/15 07:47:13 ariels Exp $
 * $Log: align_prm.h,v $
 * Revision 1.9 1998/06/15 07:47:13 ariels
 * New fields for paging of log_likelihood tables.
 *
 * Revision 1.8 1998/04/15 15:04:33 ariels
 * Add clean_(change,del,ins) parameters to align_prm for dealing with
 * "very clean" sequences in the multiple alignment.
 *
 * Revision 1.7 1998/04/15 09:28:41 ariels
 * Add parameters to create_align_prm (and to align_prm) for "dirty" characters
 * in the multiple alignment.
 *
 * Revision 1.6 1998/01/08 13:25:36 ariels
 * Make parameter block (prm) static in align_cprof.c, rather than an
 * externally supplied set of values. Therefore create_align_prm() no
 * longer returns a value.
 *
 * Revision 1.5 1997/12/31 10:42:00 ariels
 * htable pointer in align_prm is now opaque (cast to (score_t**) in
 * align_cprof.c, elsewhere this type isn't even defined).
 *
 * Revision 1.4 1997/12/17 10:25:26 ariels
 * Moved htable (table of all log_likelihood values for "thin" profile
 * nodes) pointer to be part of the align_prm structure.
 *
 * Revision 1.3 1997/12/07 15:20:51 ariels
 * Removed "dead" entropy fields from the align_prm structure.
 *
 * Revision 1.2 1997/11/30 07:40:18 ariels
 * Added Id and ChangeLog comments.
 */

```

```

#ifdef _ALIGN_PRM_H
#define _ALIGN_PRM_H

typedef struct {
    float lg_change;
    float lg_del;
    float lg_no_change_del;

    float lg_ins;
    float lg_no_ins;

    float lg_dirty_change;
    float lg_dirty_del;
    float lg_dirty_no_change_del;

    float lg_dirty_ins;
    float lg_dirty_no_ins;

    float lg_clean_change;

```

```

    float lg_clean_del;
    float lg_clean_no_change_del;

    float lg_clean_ins;
    float lg_clean_no_ins;

    float lgo;

    void *score_pgr; /* actually, score_t **htable, but opaque */
} align_prm;

void create_align_prm(float change, float del, float ins,
                     float clean_change, float clean_del, float clean_ins,
                     float dirty_change, float dirty_del, float dirty_ins,
                     float lgo);

#define GLOBAL_MODE 0x200
#define LOCAL_MODE 0x100
#define OVERLAP_MODE 0

#define SEPARATE_HILLTOPS 1

#endif

```


Sun Aug 9 10:33:25 1998

Using for Adam Santiel

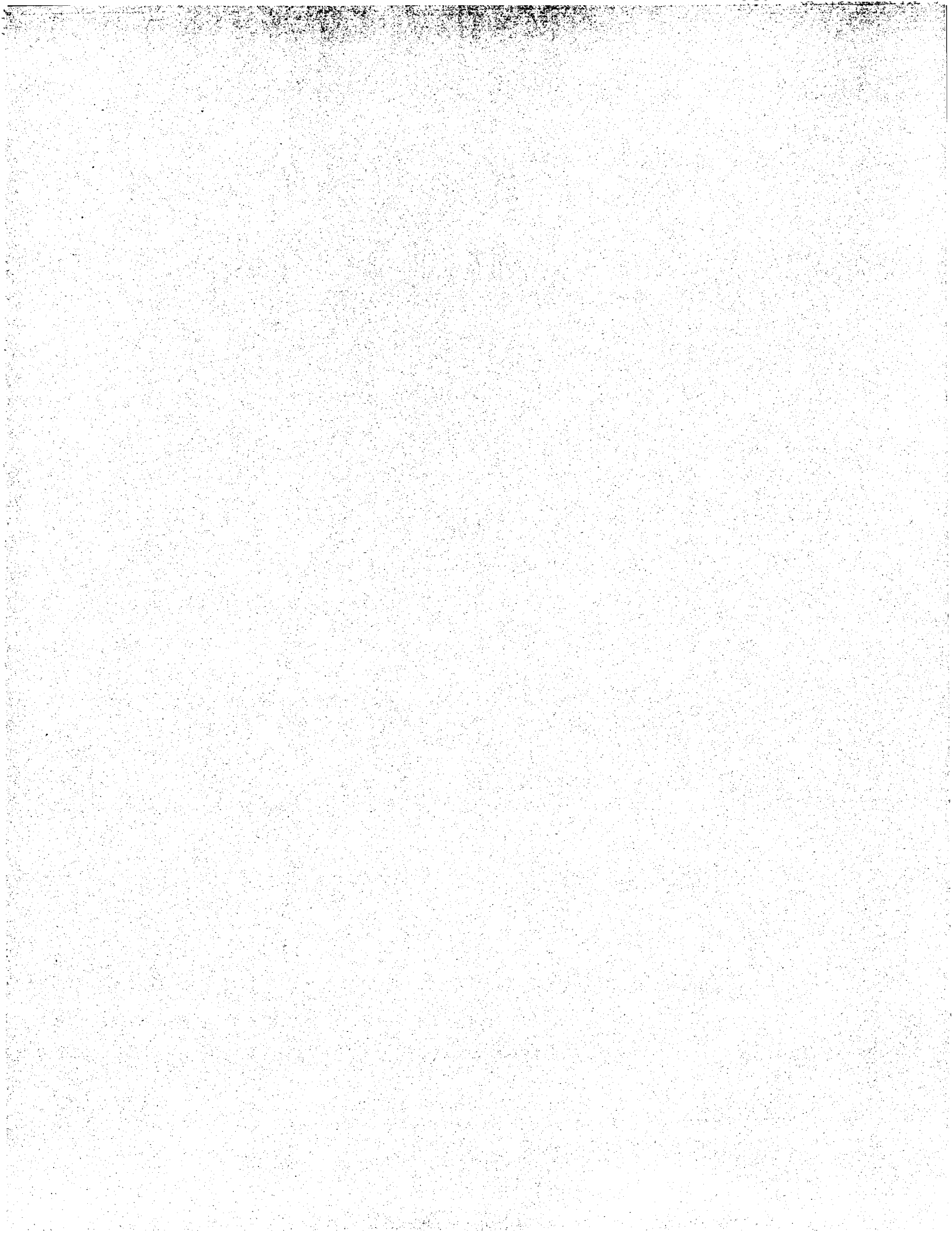
```
#ifndef ANALYZE_GRAPH_H
#define ANALYZE_GRAPH_H

#include "string.h"
#include "splice_graph.h"

int build_consensus(splice_graph *, char *consensus, int cons_map[],
                    int *nodes_order_arr);
int build_transcripts(splice_graph *graph, char *transcs[MAX_TRANSC],
                      int *trnsc_no, int **);
```

```
#endif
```

analyze_graph.h



```

/*
 * $Id: build_graph.h,v 1.9 1998/06/30 22:16:48 avner Exp $
 * $Log: build_graph.h,v $
 * Revision 1.9 1998/06/30 22:16:48 avner
 * add prototypes for cut_est_head', cut_est_tail'.
 *
 * Revision 1.8 1998/06/18 14:42:47 ariels
 * Strip "#ifdef CPROF_ALIGN" and "#ifdef IMPROVE_CPROF" lines.
 *
 * Revision 1.7 1998/02/11 11:28:47 avner
 * Needed changes after splitting build_graph to 3 different sources.
 *
 * Revision 1.6 1998/01/12 11:50:54 eval
 * Fixing warning of compilations
 *
 * Revision 1.5 1998/01/07 12:48:32 ariels
 * Added splice_node_container structure.
 *
 * Revision 1.4 1997/12/29 16:06:47 eval
 * Fixing update point_list_after_unify.
 *
 * Revision 1.3 1997/12/29 07:54:37 eval
 * Adding prototype for update_ponints_list_* (for rp mode).
 *
 * Revision 1.2 1997/11/30 07:45:29 ariels
 * Added Id and ChangeLog comments.
 *
 */

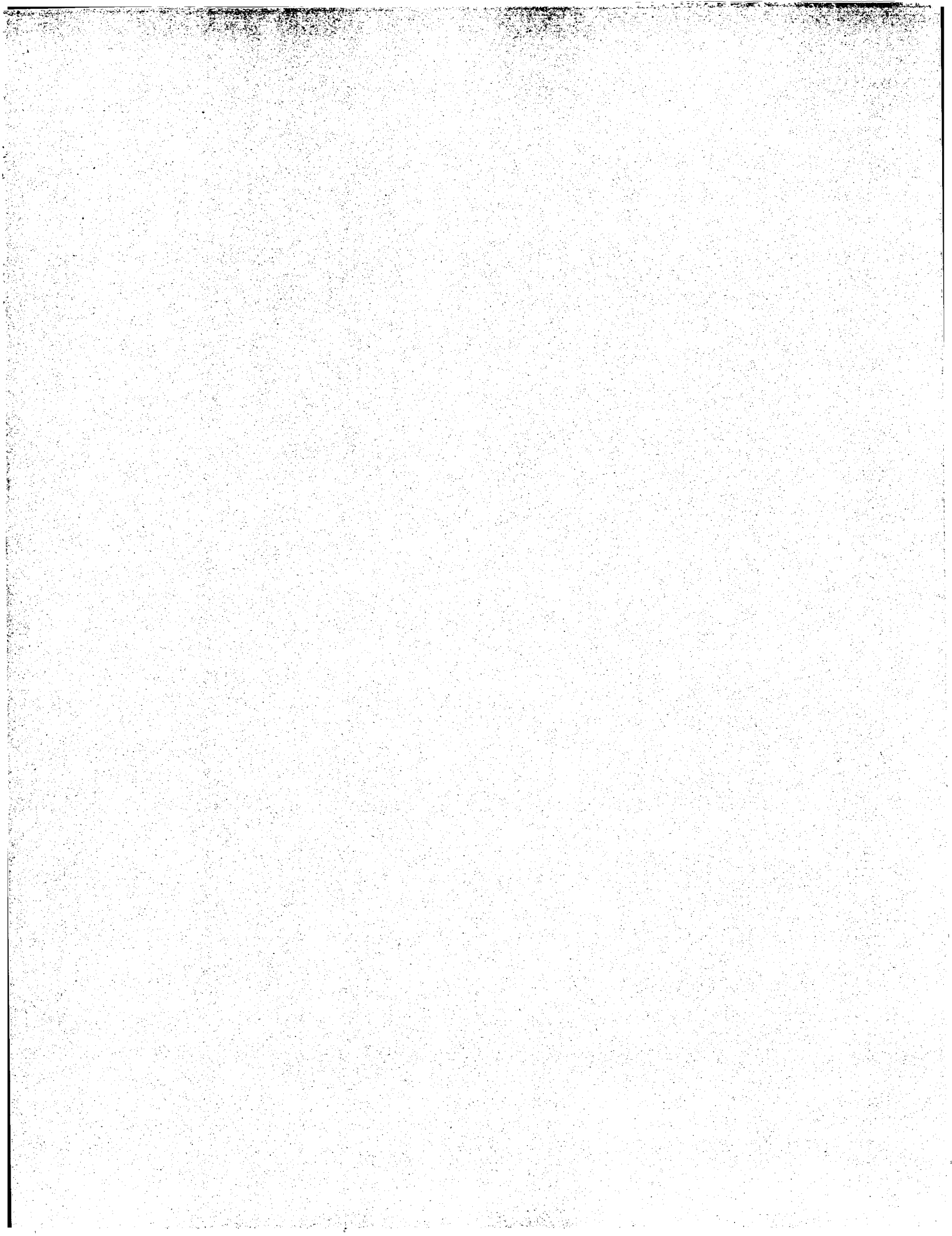
#ifdef BUILD_GRAPH_H
#define BUILD_GRAPH_H

#include <stdio.h>
#include "est_table.h"
#include "general.h"
#include "splice_graph.h"
#include "olap_graph.h"
#include "hilltops.h"
#include "custody_zones.h"

splice_graph *build_graph();
int update_splice_graph(splice_graph *graph, cprof entering_profile,
                        align_data *czm);
splice_graph *process_estts();

void cut_est_head(int est_idx, int est_head_cutpoint, splice_graph *graph);
void cut_est_tail(int est_idx, int est_tail_len, splice_graph *graph);
#endif

```



```

/* cprof.h -- Header file for "counted-profile" data structure
*
* Currently only genetic profiles are supported, i.e. profiles with 4
* letters (A,C,G,T) and a gap. The gap is always numbered 0. It
* should be relatively easy to change the alphabet to something else
* (famous last words...).
*
* Profiles are stored as pairs of dynamically allocated arrays. The
* _node_ array holds counts of the letters at each position; the
* _link_ array holds the number of links between two nodes. There
* is always one more link than there are nodes, thus link[i] comes
* between node[i-1] and node[i]. The first and last links should
* hold the number of links leading into and out of the profile. For
* some purposes this should be 0; use the LINK0 macro to get this
* behaviour.
*
* ariels@compugen 9/10/97.
*/

/* $Id: cprof.h,v 1.17 1998/05/04 08:02:06 ariels Exp $
* $Log: cprof.h,v $
* Revision 1.17 1998/05/04 08:02:06 ariels
* Add cprof_ids() (which see)
*
* Revision 1.16 1998/04/21 09:17:47 eval
* Add functions cprof_first_clean and cprof_last_clean
*
* Revision 1.15 1998/04/15 15:06:13 ariels
* Add very_clean fields to cprof for dealing with "very clean" sequences
* in the multiple alignment.
*
* Revision 1.14 1998/04/15 09:22:22 ariels
* New prototype for new cprof_node_char().
*
* Revision 1.13 1998/04/15 07:13:43 ariels
* Fix BUG (!) in CPROF_IS_BLANK due to mistaken precedence.
*
* Revision 1.12 1998/04/14 09:42:06 ariels
* New format for multiple alignment to allow "dirty" positions.
* New routine cprof_init_dirty() allows sequences with dirty tails.
*
* Revision 1.11 1998/04/08 15:26:22 ariels
* Unify 1.10.1 branch (actually identical with it); use with cprof.c
* revision >= 1.15 and align_cprof.c rev. >= 1.19.
*
* Revision 1.10.1.1 1998/02/23 14:29:35 ariels
* Klugey 'N'-counting code.
* Count everything in the cprof twice, once with N's transformed to
* random letters (for alignment) and once without the N's.
*
* Revision 1.10 1998/02/22 07:15:41 ariels
* Add cprof_get_seq_first_location function to return position of
* beginning of a sequence's pass from a cprof.
*
* Revision 1.9 1998/02/16 08:11:34 ariels
* Keep track of sequence identifiers in the multiple alignment.
*
* Revision 1.8 1998/01/27 09:28:36 ariels

```

```

* Prototype cprof_seq_ptr and cprof_count.
*
* Revision 1.7 1998/01/26 09:19:48 ariels
* Add WIDTH and SEQ fields to cprof structure, which contain the new
* multiple-alignment representation.
*
* Revision 1.6 1998/01/11 14:00:05 ariels
* Add cprof_calc_id_sim() to calculate the %age of identity and
* similarity in a given alignment. See code for exact definition of
* identity and similarity in cprof alignments.
*
* Revision 1.5 1998/01/08 13:37:18 ariels
* Prototype & documentation fixes.
*
* Revision 1.4 1997/12/31 10:37:28 ariels
* Removed 'cost' field from profiles; instead of it are a pair of static
* variables in align_cprof.c which hold the location scores for the pair
* of profiles being aligned.
*
* Removed the dead function cprof_shorten().
*
* Revision 1.3 1997/12/17 10:24:21 ariels
* Added 'score' field to profiles (used to speed up alignments)
*
* Revision 1.2 1997/11/30 07:47:42 ariels
* Added id and ChangeLog comments.
*
*/

#ifndef _CPROF_H
#define _CPROF_H

#include <stdio.h>

/* Number of distinct letters, _including gaps_ (which are always 0) */
#define N_PROFILE_LETTERS 5

/* Names of the bases; 0 is a gap! */
extern char *bases;

/*
* A counted profile consists of _nodes_ separated by _links_. Each
* node counts the letters which appear at that position; each link
* counts the number of links actually found between 2 adjacent nodes.
* This information is not redundant! (Consider, e.g., placing 2
* known sequences one after the other; there is no link between
* them).
*/

typedef int cprof_node[N_PROFILE_LETTERS];
typedef int cprof_link;
typedef int identifier_t;

typedef struct cprof_s {
    unsigned int len; /* Length of profile */
    cprof_node *node; /* len elements */
    cprof_link *link; /* len+1 elements */
}

```



```

/* --- Multiple alignment --- */
int width;
identifier_t *id;
int *pos;
short *very_clean;
unsigned char *seq;

cprof_node *realnode;
} *cprof;

/* For klugey gbl05 fix */

/* Constants and macros for storing in the multiple sequence alignments.
 *
 * 8 bits are more than enough room to store which letter (or gap) it is,
 * whether it came from an est, or an est's dirty end, and whether it's
 * really an N. 0 indicates that "nothing is there" (the sequence has
 * not yet started).
 *
 * High-quality sequences (RNA) can be indicated on a sequence basis, not
 * an individual location basis.
 */

#define CPROF_LETTER_MASK 0x0f
#define CPROF_LETTER 0x80
#define CPROF_N 0x40
#define CPROF_DIRTY 0x10
#define CPROF_BLANK 0x00

#define CPROF_IS_BLANK(x) ((x) & CPROF_LETTER)
#define CPROF_LETTER_NO(x) ((x) & CPROF_LETTER_MASK)
#define CPROF_IS_N(x) ((x) & CPROF_N)
#define CPROF_IS_DIRTY(x) ((x) & CPROF_DIRTY)

#define LINK0(prof, i) (((i) == 0 || (i) == (prof)->len) ? 0 : \
    (prof)->link[(i)])

/* Function prototypes for cprof.c */
int cprof_get_id_pos(cprof p, int pos, identifier_t id, int begin);
int cprof_get_seq_location(cprof p, identifier_t id, int loc);
int cprof_get_seq_first_location(cprof p, identifier_t est_id, int pos);

unsigned char *cprof_seq_ptr(cprof p, int i, int seq);
void cprof_count(cprof, int, cprof_node, cprof_node);

cprof cprof_new(unsigned int);
cprof cprof_init(char str[], identifier_t id, short very_clean);
cprof cprof_init_dirty(char str[], identifier_t id,
    int clean_start, int clean_end);
void cprof_free(cprof prof);

/* unify p2 into p1 */
void cprof_update(cprof p1, cprof p2, int start1, int start2,
    char *str, int flip);

/* Cut off head (tail) of profile. */

```

cprof.h

```

void cprof_cut_head(cprof, int);
void cprof_cut_tail(cprof, int);

/* Split profile, changing it to its head and returning its tail */
cprof cprof_split(cprof, int);

/* Return new copy of profile */
cprof cprof_dup(cprof);

/* Glue secondary before or after primary */
enum glue_mode {e_before, e_after};
void cprof_glue(cprof primary, cprof secondary, enum glue_mode flag, int link);

/* Return a subsequence of a profile */
cprof cprof_segment(cprof p, int start, int end);

/* Defined elsewhere (depends on model, e.g. maximal-likelihood), but
 * standardly prototyped here.
 */
/* Return a string representing a final assembly */
char *cprof_compute_assembly(cprof p, int ignore_gaps,
    int probable_mistakes[], int corrected[]);

char cprof_node_char(cprof, int loc, int *is_err, int *is_corrected);
void cprof_print(cprof);
int cprof_num_print(FILE *, char *, cprof, int inclusive, int exclusive);
/*void cprof_print_node(cprof_node, int);*/
void cprof_print_node(cprof_node);

void cprof_calc_id_sim(cprof p1, cprof p2, int start1, int start2,
    char *str, double *id_percent, double *sim_percent);
int cprof_first_clean(cprof p);
int cprof_last_clean(cprof p);
int cprof_ids(cprof p, identifier_t *ids, int sz);

#endif /* defined(_CPROF_H) */

```

cprof.h

Using for Adam Sartiell Sun Aug '9 10:33:26 1998

```

/*
 * cprof_align.h -- datatype returned from cprof 6:18 alignments
 *
 * Heavily based on align_data.h!
 *
 * ariels@compugen 19/10/97
 */

/*
 * $Id: cprof_align.h,v 1.4 1998/01/08 13:37:39 ariels Exp $
 * $Log: cprof_align.h,v $
 * Revision 1.4 1998/01/08 13:37:39 ariels
 * Prototype cprof_alignment.
 *
 * Revision 1.3 1997/12/31 10:38:53 ariels
 * Correct type for 'score' field in struct cprof_align_s.
 *
 * Revision 1.2 1997/11/30 07:49:32 ariels
 * Added Id and ChangeLog comments.
 *
 */

#ifndef _CPROF_ALIGN_H
#define _CPROF_ALIGN_H
#include "cprof.h"

#define ALIGN_MATCH      ''
#define ALIGN_GAP1      '1'
#define ALIGN_GAP2      '2'
#define ALIGN_LGAP1     'a'
#define ALIGN_LGAP2     'b'
#define ALIGN_ALT       '#'

typedef struct cprof_align_s {
    struct cprof_align_s *next;
    /* int node_id; */
    float score;
    int start1, end1;
    int start2, end2;
    /* int prefix, suffix;
    char *str;
    */ *cprof_align;
    /* Location of this alignment in 1st profile */
    /* Location of this alignment in 2nd profile */
    /* is the left/right of node in alignment? */
    /* String of states for this alignment */
} *cprof_align;

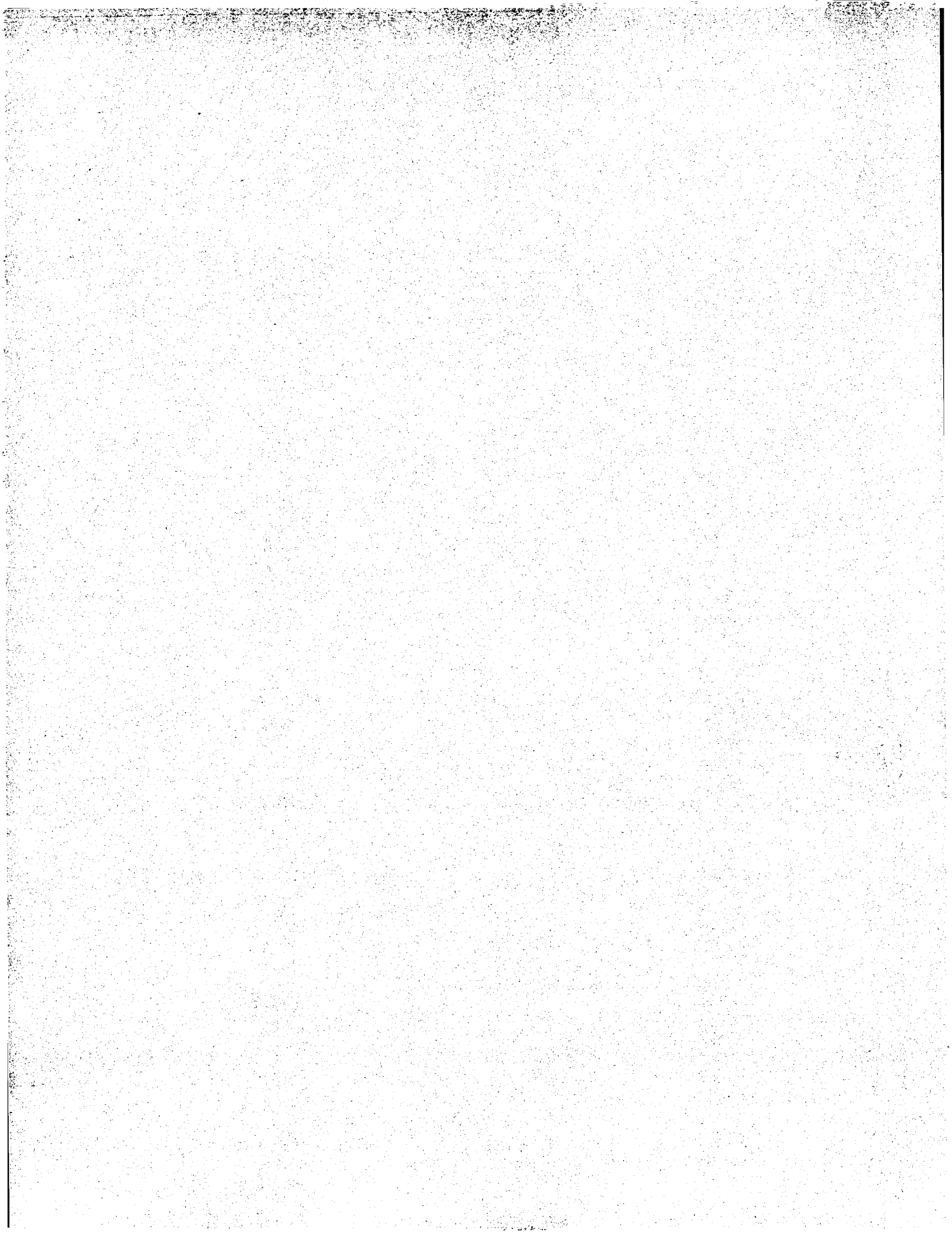
void cprof_align_free(cprof_align);

cprof_align cprof_alignment(cprof, cprof, int);

#endif /* ! defined(_CPROF_ALIGN_H) */

```

cprof_align.h



```

/*
 * $Id: custody_zones.h,v 1.5 1998/06/29 12:10:34 eyal Exp $
 * $Log: custody_zones.h,v $
 * Revision 1.5 1998/06/29 12:10:34 eyal
 * Shift prototypes to .h file (where they belong).
 *
 * Revision 1.4 1998/06/18 14:42:47 ariels
 * Strip "#ifdef CPROF_ALIGN" and "#ifdef IMPROVE_CPROF" lines.
 *
 * Revision 1.3 1998/02/14 17:27:36 avner
 * Change prototype for 'adl_to_czm' when CPROF_ALIGN (gets entering profile)/.
 *
 * Revision 1.2 1997/11/30 08:08:37 ariels
 * Added Id and ChangeLog comments.
 *
 */

#ifdef CUSTODY_ZONES_H
#define CUSTODY_ZONES_H

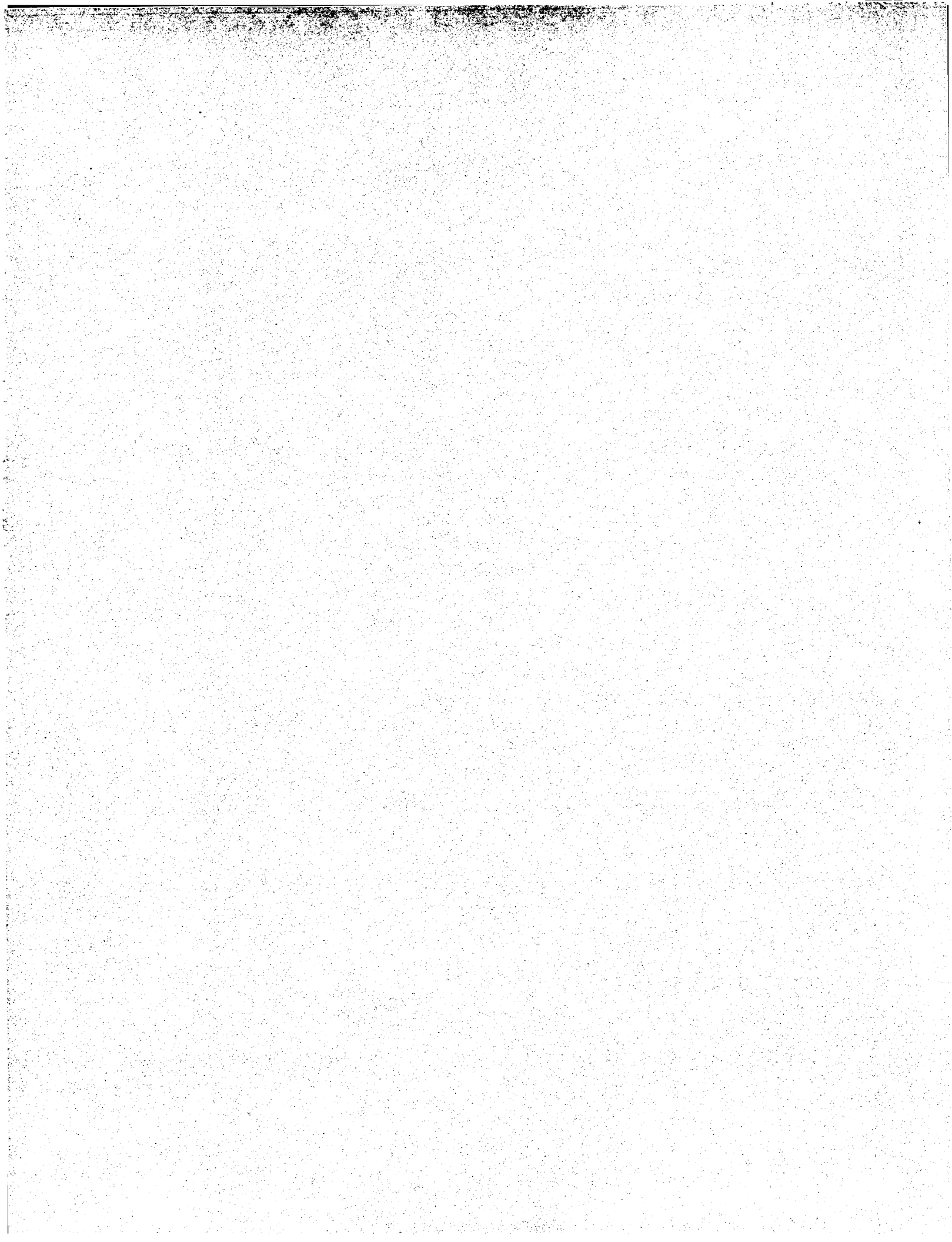
#include <stdio.h>
#include <string.h>
#include "splice_graph.h"
#include "cprof.h"
#include "align_data.h"
#include "general.h"

#define DUMMY -1
#define FREE_SEG -1 /* some constant that can not be a node number */

align_data *align_data_new (align_data *old, int start_est, int end_est);
void align_data_adl_to_czm (align_data *adl, int len_est, cprof entering_profile);
void align_data_list_print (align_data *p);
void align_data_list_free (align_data *p);
void copy_align_data (align_data *from, align_data *to);
void czm_print (align_data *czm);
void czm_shift_right (align_data *czm, int shift);
void czm_shift_left (align_data *czm, int shift);
void czm_shift_right_of_same_node (align_data *czm, int id, int shift);
void cut_align_str (align_data *czm, int offset);
align_data *new_adl_to_czm (align_data *adl, int len_est, cprof entering_profile,
                           splice_graph *graph);

#endif

```



```

/* $Id: dp.h,v 1.2 1997/11/30 08:09:52 ariels Rel $
 * $Log: dp.h,v $
 * Revision 1.2 1997/11/30 08:09:52 ariels
 * Added Id and ChangeLog comments.
 *
 */

#ifndef DP_H_
#define DP_H_

#define MAX_FULL 1200
#define MAX_LEN 250000
#define MAX_SEC 2500
#define TANDEM_GAP 20
#define EPSILON 0.00001

#define BEST -1
#define CORNER -2

/* Constants for six18_alignment mode */
#define ONE_HILLTOP 0
#define SEPARATE_HILLTOPS 1
#define LINEAR_SPACE 2
#define HILLTOP_MASK 0xff
#define OVERLAP_MODE 0
#define LOCAL_MODE 0x100

#include "hilltops.h"

void init_matrix (int match, int mismatch, int tA, int tB);

hilltop *Hilltops (char in_x[], char in_y[], int mode, int cutoff, int bound);
double smith_waterman (char in_x[], char in_y[]);
void global_alignment (char in_x[], char in_y[], int loop_end, hilltop **ht);
void get_alignment (char in_x[], char in_y[], hilltop *ht, int where);
void linear_alignment (char in_x[], char in_y[], hilltop *ht, int bs, int es);
hilltop *get_six18_alignment (char in_x[], char in_y[], int mode);
hilltop *band_sw (char x[], char y[], int x0, int y0, int width);

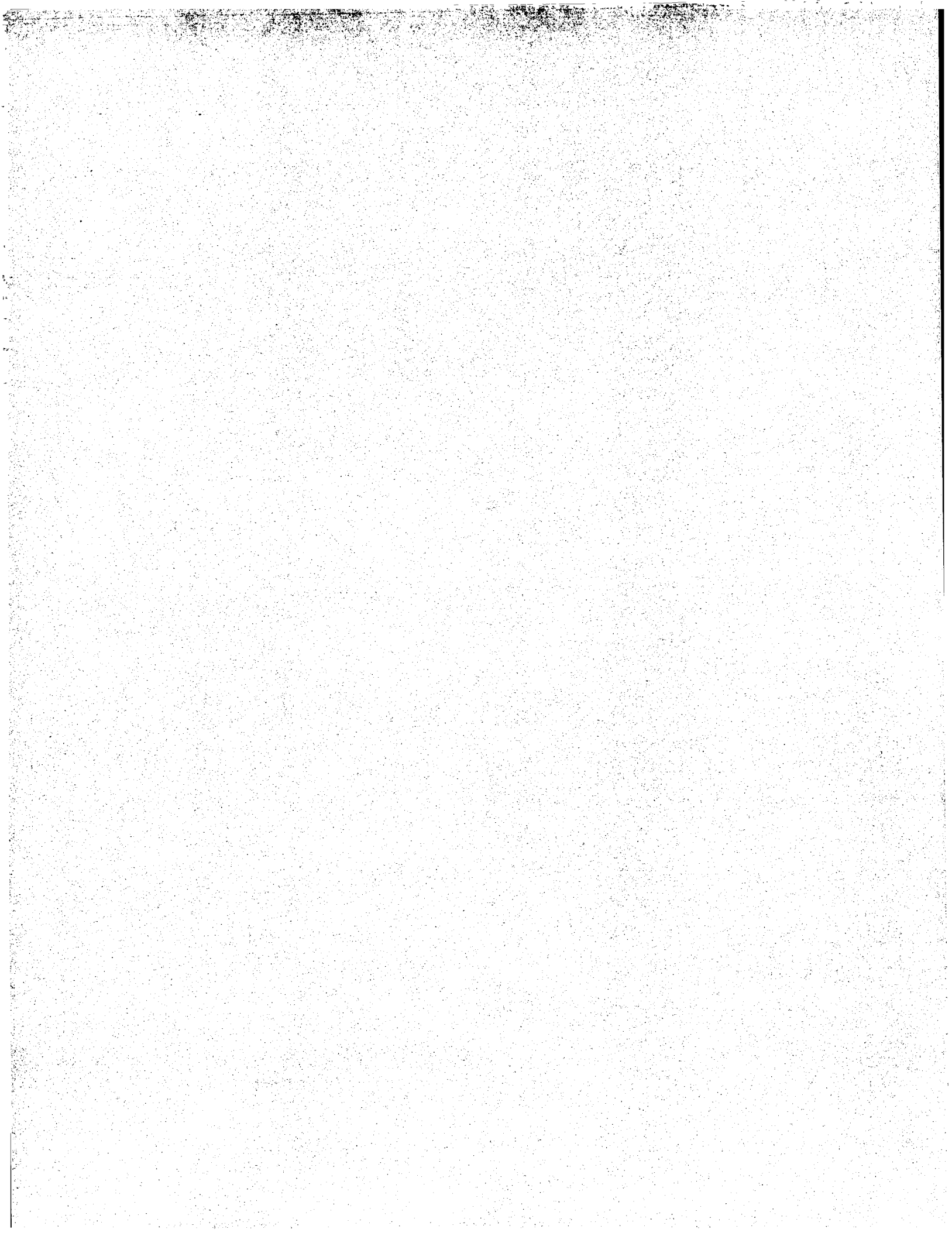
void print_alignment (hilltop data);

/* Structure definitions for hilltops */
typedef struct mem {
    int s;
    unsigned int xy;
} mem;

typedef struct mmode {
    mem match, ins, del;
} mmode;

#endif

```



Sun Aug 9 10:33:27 1998

Using for Adam Sattel

```

/* Include file for error reporting
*/
/*
 * $Log: error.h,v $
 * Revision 1.2 1998/02/22 17:11:44 artels
 * Add "delimited" one-line reports.
 *
 * Revision 1.1 1998/01/20 09:31:35 artels
 * Initial revision
 */

#ifndef _ERROR_H
#define _ERROR_H

#include <stdio.h>

typedef struct err_list_s {
    char *msg;
    struct err_list_s *next;
} *err_list;

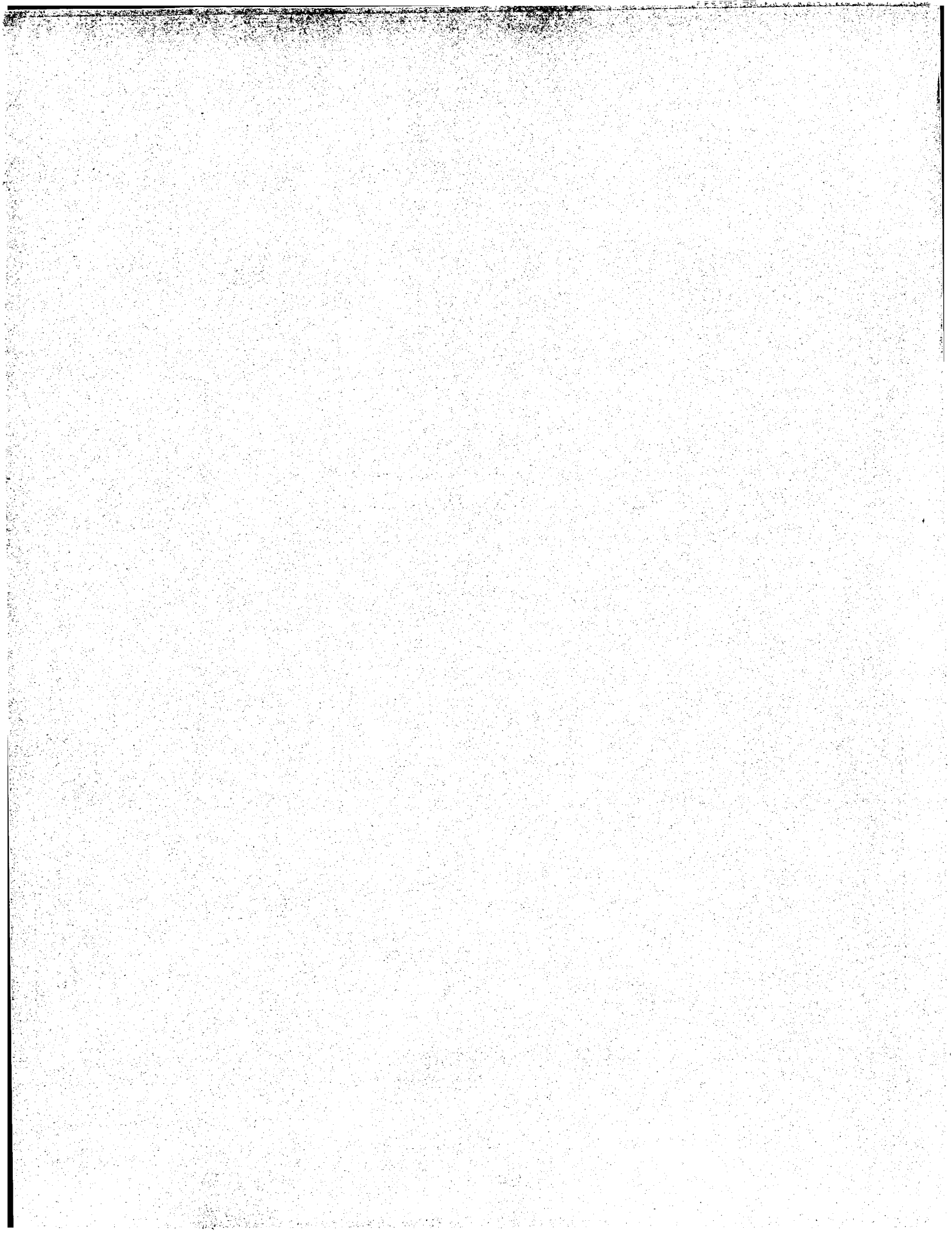
void err(char *fmt, ...);
int ret_err(char *fmt, ...);

void record_warn(char *fmt, ...);
void record_err(char *fmt, ...);
int warned(void);
int erred(void);
void report_warn(FILE *fp, char *prefix);
void report_warn_online(FILE *fp, char *delim);
void report_err(FILE *fp, char *prefix);
void report_err_online(FILE *fp, char *delim);
void clear_warn(void);
void clear_err(void);

#endif /* defined(_ERROR_H) */

```

error.h



```

/*
 * $Id: est_table.h,v 1.12 1998/06/29 12:16:51 avner Exp $
 * $Log: est_table.h,v $
 * Revision 1.12 1998/06/29 12:16:51 avner
 * added prototype for 'est_table__next_in_variant'.
 *
 * Revision 1.11 1998/06/15 12:13:20 eval
 * Add support for hyper_edge_node list for all the ests, and function which che
ck it
 *
 * Revision 1.10 1998/05/11 11:53:30 eval
 * Add function est_table__verify_est_path
 *
 * Revision 1.9 1998/05/04 12:35:59 eval
 * Add field and get/set functions component
 *
 * Revision 1.8 1998/04/13 10:36:37 eval
 * 1. change feild node_path from short to int..
 * 2. add function est_table_switch_node_path_hyper_edge
 *
 * Revision 1.7 1998/03/30 08:52:13 eval
 * Add hyper_edge and hyper_edge_len fields to est - this is the path
of the est built with the cprof info.
 *
 * Revision 1.6 1998/03/17 16:37:17 avner
 * support for clone-info and uncleaning.
 *
 * Revision 1.5 1998/03/08 21:16:32 avner
 * add two fields in type 'est': 'start_in_cons' and 'end_in_cons' for
the use of clone-length prediction mechanism.
 *
 * Revision 1.4 1998/02/11 16:24:29 ariels
 * Support 'contig' field for (separate) full-cluster flipper program.
 *
 * Revision 1.3 1998/01/14 10:04:17 eval
 * Add feild test_case and set get functions
 *
 * Revision 1.2 1997/11/30 08:12:00 ariels
 * Added Id and ChangeLog comments.
 */

#ifndef EST_TABLE_H
#define EST_TABLE_H

#include <string.h>
#include "general.h"
#include "dp.h"
#include "io.h"
#include "hyper_graph.h"

typedef struct olap_edge {
    int est_id;
    int st1,st2,end1,end2;
    int inverted;
    struct olap_edge *next;
} olap_edge;

typedef struct est_ {
    rich_fasta_seq_ptr seq_data; /* sequence data obtained by rich fasta */

```

```

char key[MAX_KEY_LEN+1];
int in_degree;

int invert_algn_color;
int ordinal;
int variant_no;
olap_edge *olaps;

int clone_pair_idx;
int *node_path;
int *hyper_edge_path;

int hyper_edge_len;
hyper_edge_node *hyper_edge;

int ximeric;
int test_case;
int contig;
int start_cleaned;
int end_cleaned;
int start_in_cons;
int end_in_cons;
int component;
} est;

typedef struct table_ {
    int size;
    est *arr[MAX_CLUSTER_SIZE];
    int rna_num,est_num;
    char **keywords[5];
    table_type;

    int est_table__is_rna(int);
    int est_table__is_est(int);
    int est_table__rna_num();
    int est_table__est_num();
    void est_table__create_keywords();
    void est_table__keywords(int,int);
    void est_table__add(rich_fasta_seq_ptr);
    char *est_table__seq(int idx);
    char *est_table__original_seq(int idx);
    char *est_table__cleaned_seq(int idx);
    rich_fasta_seq_ptr est_table__seq_data(int idx);
    char *id(int idx);
    void est_table__clean();
    void est_table__invert_seq(int idx);
    int est_table__is_inverted(int idx);
    void est_table__put_ordinal(int idx,int ordinal);
    int est_table__get_ordinal(int idx);
    void est_table__put_inverse_color(int idx,int clr);
    int est_table__get_inverse_color(int idx);
    int est_table__get_deg(int idx);
    void est_table__increment_deg(int idx);
    char *id(int idx);
    int est_table__len(int);
    int est_table__size();
    void est_table__print(FILE *fp);
    int look_up(char *key);
    void est_table__add_edge(int idx1,int idx2,int st1,int st2, int endl, int end2)

```

```

;
olap_edge *est_table__neighbors(int idx1,int idx2);
olap_edge *est_table__neighbor_list(int idx);
void est_table__put_variant(int est_idx,int variant);

void est_table__pair_clones(void);
void est_table__set_node_path(int idx, int *path, int size);
int est_table__node_path_pos(int idx, int pos);
char* est_table__get_clone(int est);
int est_table__any_ximeric(void);
void est_table__set_ximeric(int idx);
int est_table__is_ximeric(int idx);
void est_table__test_case(int idx);
void est_table__set_test_case(int idx);
int est_table__contig(int idx);
void est_table__set_contig(int idx, int contig);
void est_table__mark_alignment_range(int idx,int start, int end);
int est_table__get_start_cleaned(int idx);
int est_table__get_end_cleaned(int idx);
int est_table__get_start_in_cons(int idx);
int est_table__get_end_in_cons(int idx);
char *est_table__clone_name(int idx);
int est_table__begin_clean(int idx);
int est_table__end_clean(int idx);
int est_table__clone_length(int five_p,int three_p);
int *est_table__get_node_path(int est);
void est_table__get_hyper_edge_path(int idx, int *hyperedge);
hyper_edge_node *est_table__get_hyper_edge(int idx);
void est_table__set_hyper_edge(int idx, hyper_edge_node *hyperedge);
int est_table__get_hyper_edge_len(int idx);
void est_table__set_hyper_edge_len(int idx, int len);
void est_table__set_component(int idx, int component);
int est_table__get_component(int idx);
int est_table__verify_est_path(int idx);
int est_table__next_in_variant(int id,int start_seq);

#endif

```

Listing for Adam Sartiel Sun/Aug 9:10:33:27 1998

```
/*
* $Id: general.h,v 1.7 1998/06/25 08:54:42 ariels Exp $
* $Log: general.h,v $
* Revision 1.7 1998/06/25 08:54:42 ariels
* Add "maximal cprof memory" parameters giving maximal values of
* maximal cprof memory.
*
* Revision 1.6 1998/04/24 12:46:17 avner
* defined ASSEMBLY_MODE and FLIPPER_MODE.
*
* Revision 1.5 1998/04/18 21:47:34 avner
* Change 'MAX_TRANS' from 20 to 100.
*
* Revision 1.4 1998/02/22 17:17:25 ariels
* Use a REAL '%' sign in format strings.
*
* Revision 1.3 1998/02/22 11:11:44 avner
* Change 'INTERSECTION_MESSAGE' to 'INTERSECTION_MESSAGE', and make it a
* format string that can take two variables of type double.
*
* Revision 1.2 1997/11/30 08:12:25 ariels
* Added Id and ChangeLog comments.
*
* */
#endif GENERAL_H
#define GENERAL_H

/* #define DESCENDING_ORDER */

#include "parameters.h"

#define MAX_CPROF_ALIGN_MAX_MEM (100*(1<<20))
#define MAX_CPROF_ALIGN_MAX_TBL_SZ (100*(1<<20))

#define LINE_SIZE 5000
#define UNIX_FNAME_LEN 100
#define CLUSTER_NAME_SIZE 50
#define MAX_CLUSTER_SIZE 10000
#define MAX_KEY_LEN 40
#define min(a,b) ((a>b)?b:a)
#define max(a,b) ((a>b)?a:b)
#define MAX_TRANS 100
#define MAX_TRANS_LEN 30000

#define DIRECTED_CYCLE_MSG "a (directed) cycle was detected in the splice-graph"
#define INTERSECTING_ALIGNMENTS "big intersection (%.0f%%<-->%.0f%%) between dif
ferent alignments"
#define DISCONNECTED_MSG "disconnected"
#define INVERSION_INCONS "reverse-alignment-info inconsistent"
#define ORDER_INCONS_ALIGN "order-inconsistent alignment (maybe a repeat)"
#define ID 0
#define CLONE 1
#define TISSUE 2
#define LIBRARY 3
#define CHROMOSOME 4

/* for the distinction in shared function between the applications */
#define ASSEMBLY_MODE 0
#define FLIPPER_MODE 1
```

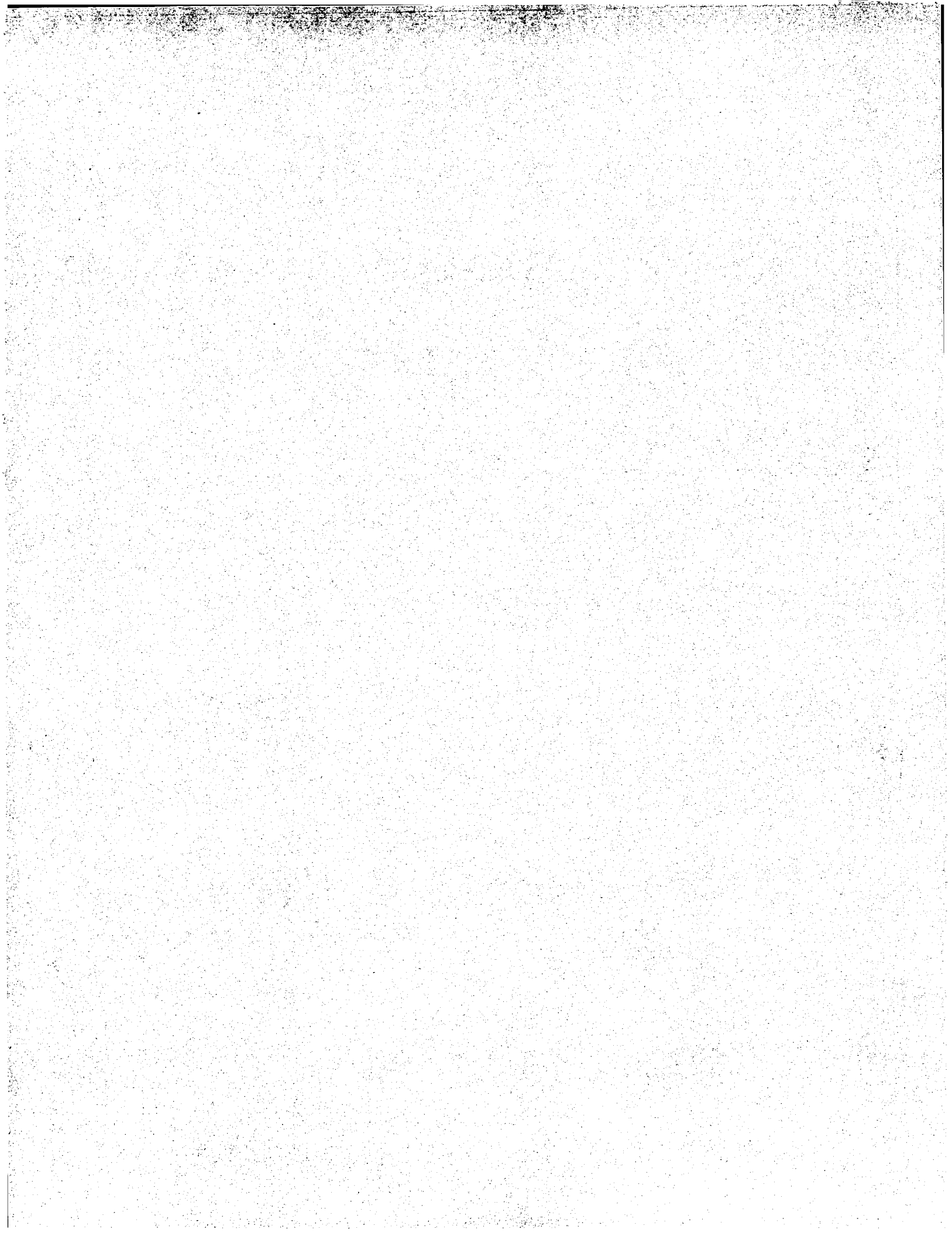
general.h

Listing for Adam Sartiel Sun/Aug 9:10:33:27 1998

#endif

general.h

A-189



```
/*
 * $Id: hilltops.h,v 1.2 1997/11/30 08:13:00 ariels Rel $
 * $Log: hilltops.h,v $
 * Revision 1.2 1997/11/30 08:13:00 ariels
 * Added Id and ChangeLog comments.
 */

#ifndef HILLTOPS_INC
#define HILLTOPS_INC

#define STRAIGHT -1
#define INVERTED -2
#define TANDEM -3
#define WRAPAROUND -4
#define UNDEFINED -5

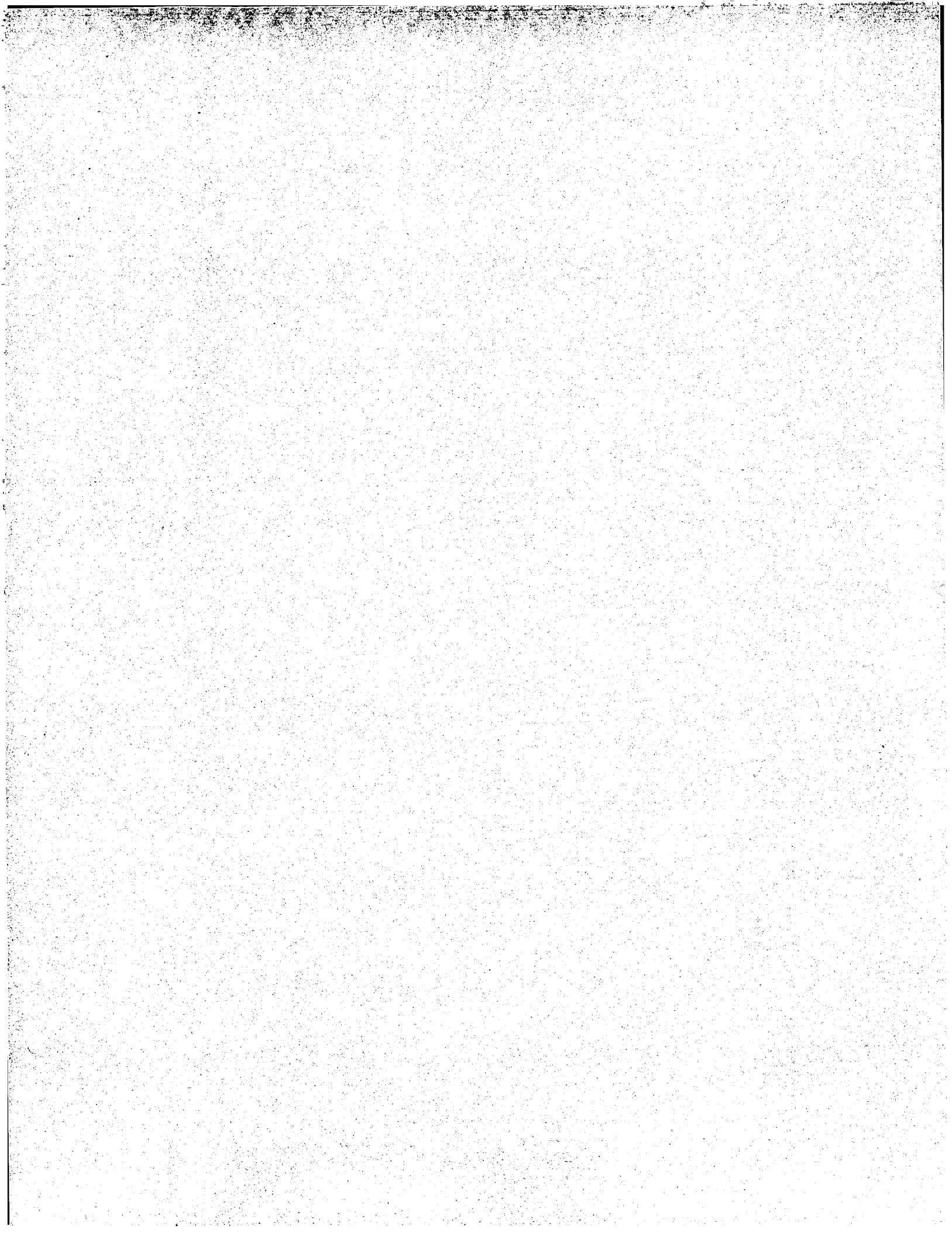
#define NAME_LEN 20

typedef struct segment {
    struct segment *next;
    int start, end;
    double score;
    int best_y;
    int x0, y0;
} segment;

typedef struct hilltop {
    struct hilltop *next;
    int type;
    char name[NAME_LEN];
    int final;
    char *x, *y;
    char *diff;
    double id_percent;
    double sim_percent;
    int len;
    int x0, y0;
    int xt, yt;
    double score;
    int area;
    segment *boundary;
} hilltop;

/* l=string, -2=inv 0 or positive = long repeat no. */
/* name of repeat */
/* flag: 1 = final alignment, 0 = under work */
/* output sequences */
/* difference between output sequences */
/* percentage of identity in the alignment */
/* percentage of similarity in the alignment */
/* length of output sequences (identical) */
/* position of beginning of alignment in input */
/* position of end of alignment in input */
/* score of match */
/* Hilltop area */
/* linked list of segments */

void create_segment (segment **obj);
void destroy_segment (segment **obj);
void destroy_segment_list (segment **obj);
void copy_segment (segment from, segment *obj);
void create_hilltop (hilltop **obj);
void destroy_hilltop (hilltop **obj);
void destroy_hilltop_list (hilltop **obj);
void copy_hilltop (hilltop from, hilltop *obj);
void update_all_hilltops (segment *seg_list, hilltop **ht_list, int x, int w);
void update_hilltop (hilltop *htp, segment seg, int x);
void merge_hilltops (hilltop *htp1, hilltop *htp2);
void trim_hilltop (char in_x[], char in_y[], hilltop *ht, int cutoff);
char *subseq (char *seq, int first, int last);
char inv (char c);
```



```
#ifndef HYPER_GRAPH_H
#define HYPER_GRAPH_H

#include "splice_graph.h"

typedef struct hyper_edge_node {
    int node_id;
    int est_start;
    int est_end;
    int cprof_start;
    int cprof_end;
} hyper_edge_node;

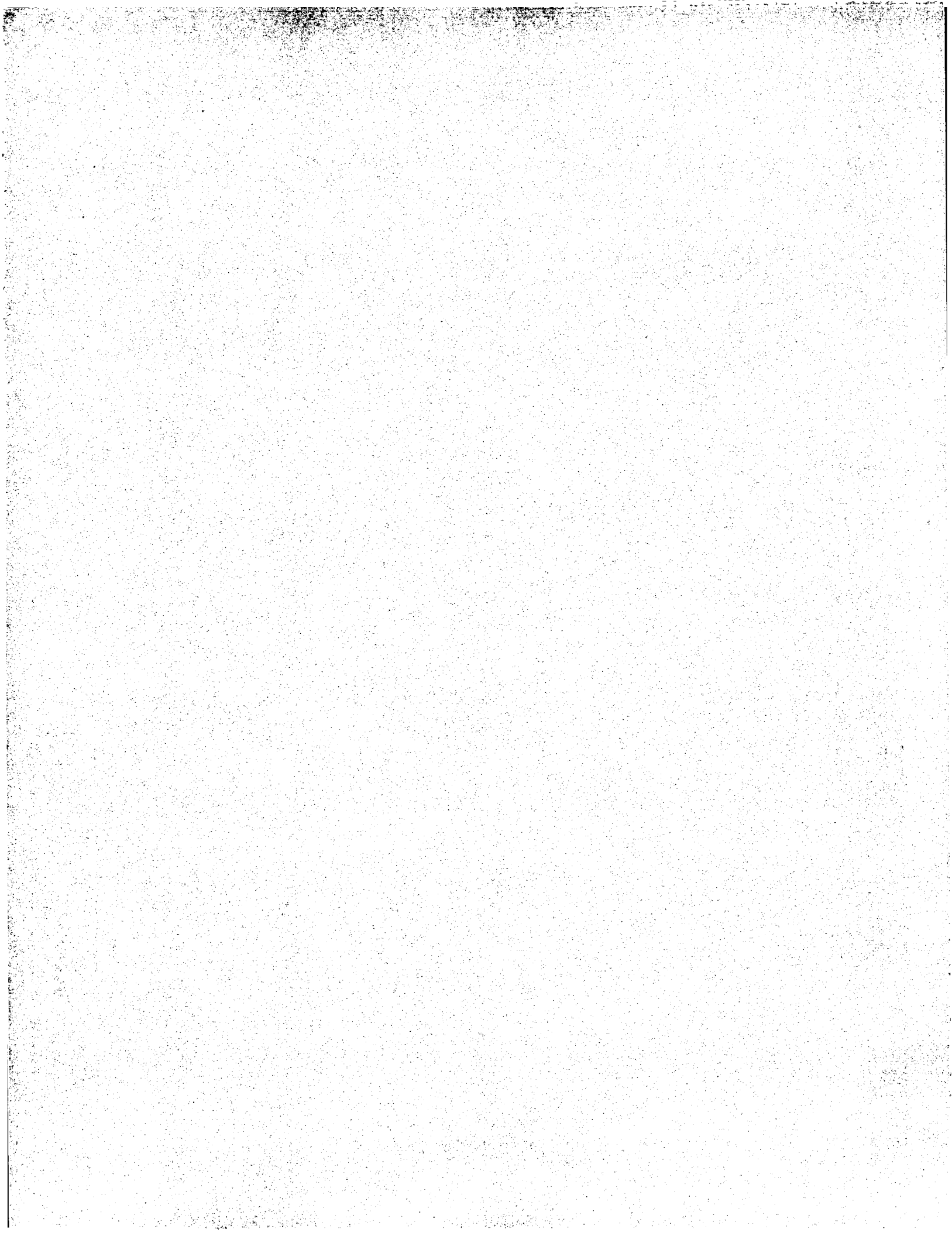
typedef struct dual_edge {
    struct hyper_edge *source;
    struct hyper_edge *target;
    int relationship;
} dual_edge;

typedef struct hyper_edge {
    int id;
    int len;
    int *nodes;
    int est;
    dual_edge neighbors[MAX_CLUSTER_SIZE];
    int out_degree;
    struct hyper_edge *contained_edges[MAX_CLUSTER_SIZE];
    int contained_num;
    int is_maximal; /* 1 iff there is no edge containing this one */
    int mark; /* hold the number of occurrence of the edge in the stack */
    splice_graph *graph;
} hyper_edge;

typedef struct hyper_graph {
    int size;
    struct hyper_edge *edge_list[MAX_CLUSTER_SIZE];
    splice_graph *s_graph;
} hyper_graph;

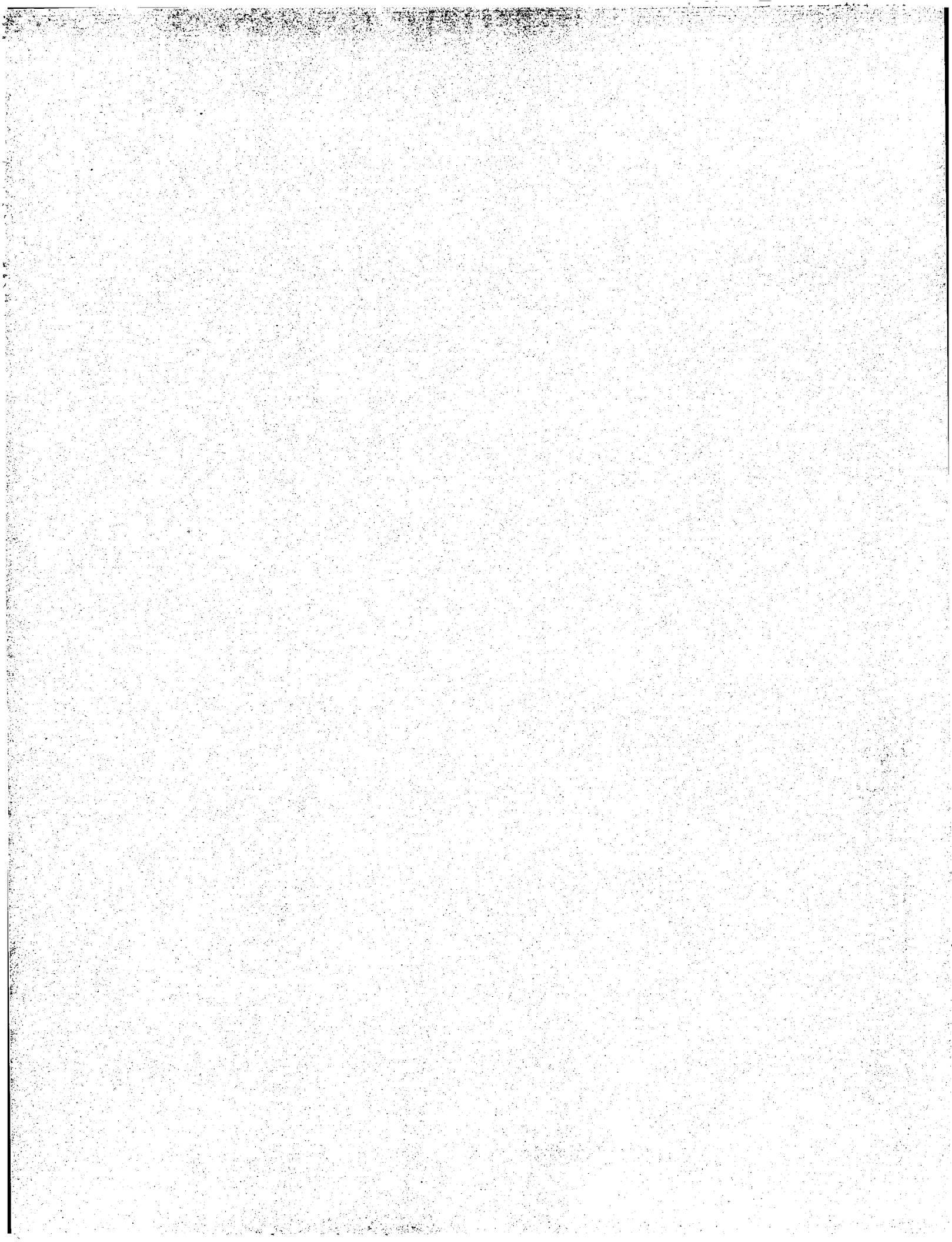
hyper_graph *build_hyper_graph(splice_graph *graph);
void hyper_graph_free(hyper_graph *graph);
hyper_edge *hyper_graph_get_edge(int idx, hyper_graph *graph);
void find_ests_paths(splice_graph *graph);

#endif
```

```
/* *****  
IO functions  
header and types  
*****  
*/  
/* $Id: io.h,v 1.13 1998/04/19 08:06:44 eyal Exp $  
* $Log: io.h,v $  
* Revision 1.13 1998/04/19 08:06:44 eyal  
* Increase the size MAX_HEADER to 1500  
*  
* Revision 1.12 1998/04/18 22:12:52 avner  
* Change prototype of 'print_alignment_genweb'.  
*  
* Revision 1.11 1998/04/12 12:47:19 eyal  
* Change map from char* to int* in write_cluster_output and print_alignment_genweb  
*  
* Revision 1.10 1998/04/12 08:38:22 avner  
* 1. introduce type hash_item for allowing tissue printing for transcripts with  
* multiplicity.  
* 2. work in progress for printing transcripts in cvview file.  
*  
* Revision 1.9 1998/04/09 17:07:51 eyal  
* add a parameter to 'read_sequences' called 'assembly_unit'. When it's 'u', the  
* unit we read here  
* is a cluster and not a contig as in the other cases.  
*  
* Revision 1.8 1998/03/17 16:38:14 avner  
* change prototype of 'print_alignment_genweb'.  
*  
* Revision 1.7 1998/02/17 16:16:43 ariels  
* Fix name of global variable cluster_no (to contig_name), since it was  
* neither CLUSTER nor NO (number)...  
*  
* Revision 1.6 1998/02/08 13:35:13 ariels  
* Separated rich-Fasta I/O routines into rf.[ch]  
*  
* Revision 1.5 1997/12/22 08:12:41 ariels  
* Change read_sequences to return (also) contig_id (cid) in the form  
* CLUSTER.CONTIG when reading GenBank >=104 format rich-Fasta input  
* files.  
*  
* Revision 1.4 1997/12/17 14:10:07 ariels  
* Added 'header' field to struct rich_fasta (stores a copy of the header  
* as it was read in).  
*  
* Revision 1.3 1997/12/17 13:02:51 ariels  
* From GenBank 104 onwards, rich-Fasta should have a "#CU" cluster  
* (formerly metacluster) field and a "#CN" contig (formerly cluster)  
* field, instead of just the old "#CU" field. Parse this field when  
* reading rich-Fasta.  
*  
* Revision 1.2 1997/11/30 08:13:33 ariels  
* Added Id and ChangeLog comments.  
*  
*/
```

```
#ifndef _IO_H_  
#define _IO_H_  
  
#include <stdio.h>  
#include "rf.h"  
#include "splice_graph.h"  
  
#define TBL_SZ 10861  
#define MAX_HEADER 1500  
  
typedef struct hash_item_s {  
    char *name;  
    int multiplicity;  
} hash_item;  
  
typedef hash_item (*tissue_hash) [TBL_SZ];  
  
typedef struct cluster_classification {  
    int num_initial_nodes;  
    int num_terminal_nodes;  
    int undirected_cycles;  
    int distinguished_ends;  
    int ximeric;  
    int volume;  
} cluster_classification;  
  
void read_sequences(fasta_file, char *, char *, char);  
  
void write_cluster_output(char **transcripts, int n, char *consensus,  
                           int consensus_len, int *node_order,  
                           int unique_order,  
                           char *contig_name, int map[],  
                           int **transc_map,  
                           tissue_hash *tissues,  
                           splice_graph *graph,  
                           hilltop **hts);  
  
void print_alignment_genweb(char *consensus, int est_idx, hilltop data, FILE *fp,  
                             int *map, char *colors, char dirty_alignment_color, char poly_a_color);  
void get_alignment_errors(hilltop *ht, int len, int *in, int *de, int *mi, int *n  
                           o);  
  
void print_alignment (hilltop data);  
  
tissue_hash new_tissue_hash(void);  
int tissue_hash_intern(tissue_hash, char *);  
void tissue_hash_print(tissue_hash, FILE *, char *, char*);  
  
#endif
```



```

/* $Id: olap_graph.h,v 1.2 1997/11/30 08:13:50 ariels Rel $
 * $Log: olap_graph.h,v $
 * Revision 1.2 1997/11/30 08:13:50 ariels
 * Added Id and ChangeLog comments.
 *
 */

#ifndef OLAP_GRAPH_H
#define OLAP_GRAPH_H

#define OLAP_PRIORITY_FACTOR 200

#include <stdio.h>
#include "general.h"

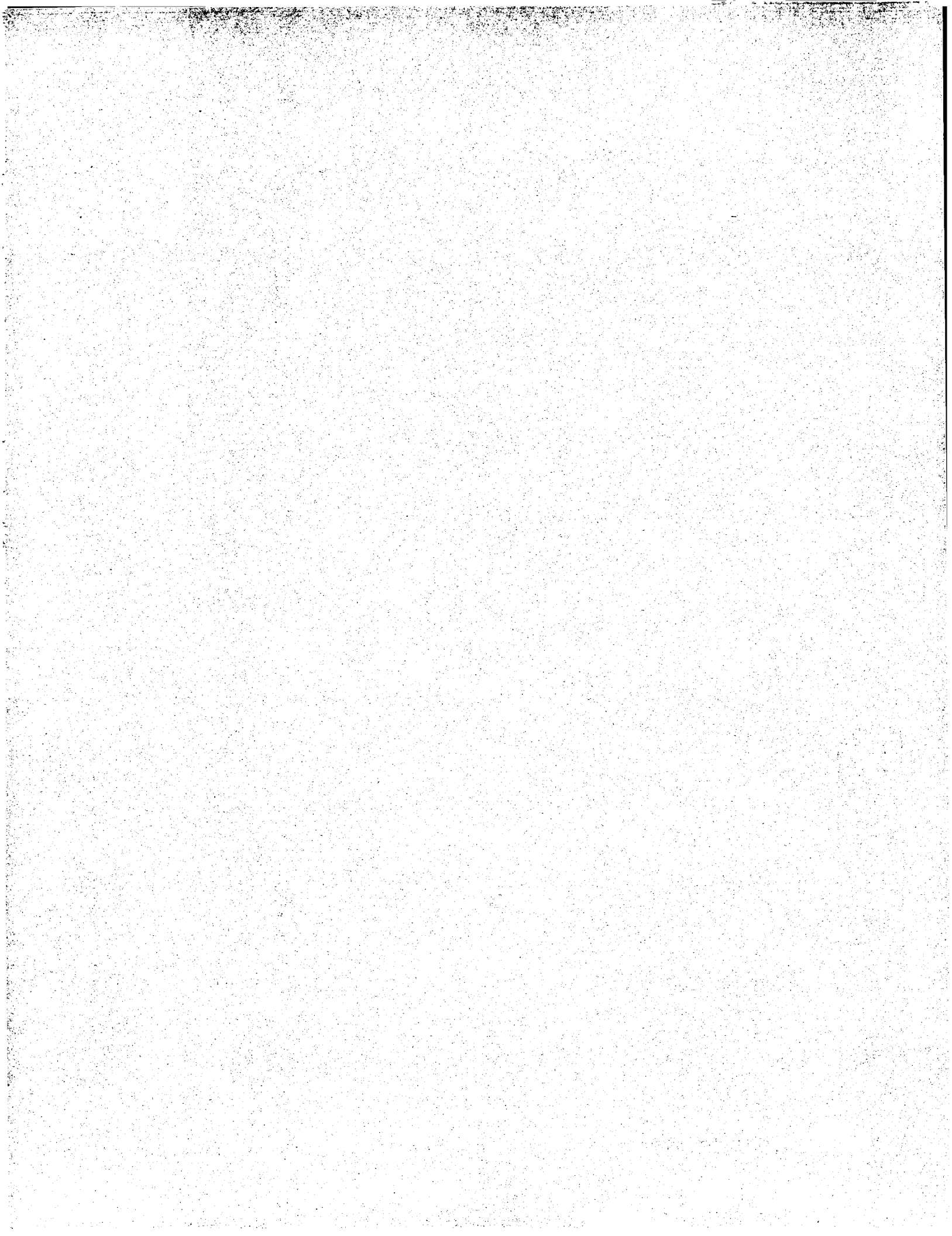
typedef struct edge_in_olap_graph_ { /* describing the alignment between i and j
 */
    int edge;
    int st1,endl,st2,endl2;
    int left_in_algn,right_in_algn;
    int inverted_algn;
    int predecessor;

    int left_contain;

    } edge_in_olap_graph;

#endif
/* there is some alignment (the info
/* later is relevant just then)
/* the left(right) of either i or j is
/* i ~!j
/* i in the alignment
/* j has a substantial fraction priot
/* to i
/* alignment starts (left side)
/* together and i is longer
/*
/*

```



Sun Aug 9 10:33:28 1998

Using /on Adam Santei/

Page

1

```
/* $Id: parameters.h,v 1.3 1998/02/08 08:30:39 avner Exp $
 * $Log: parameters.h,v $
 * Revision 1.3 1998/02/08 08:30:39 avner
 * This file turns to be parameters file ONLY for the concept of dubious
 * alignment. At this stage the business of these type of alignments does not
 * seem important enough to introduce those parameters as 'flexible' parameters.
 *
 * Revision 1.2 1997/11/30 08:14:19 ariels
 * Added Id and ChangeLog comments.
 *
 */

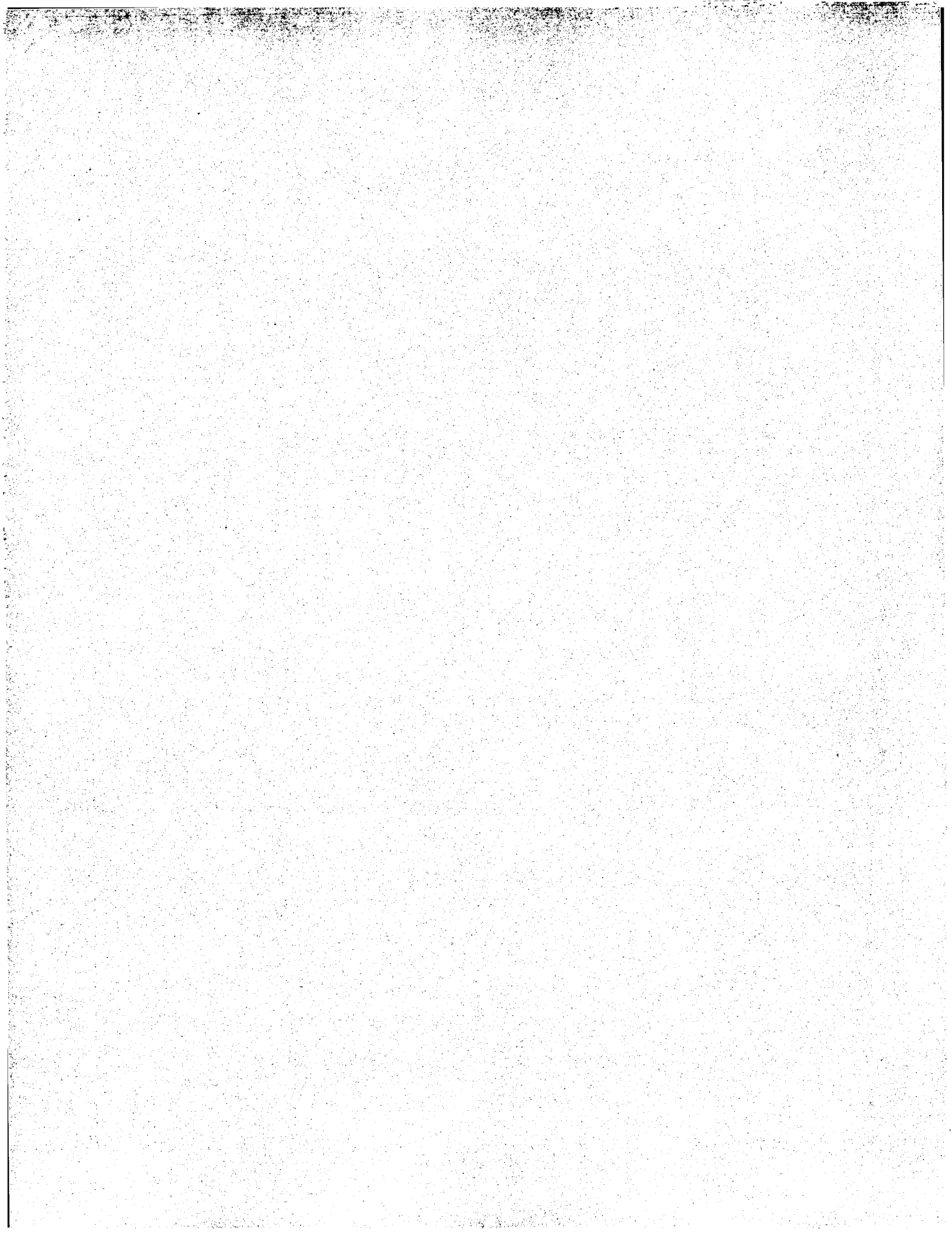
#ifdef PARAMETERS_H
#define PARAMETERS_H

#define HIGH_LEN_BOUND 20 /* alignment above this considered 'certain' */

#define GLUE_LOW_END -10 /* for a dubious alignment to survive it has
 * to be no more than -GLUE_LOW_END away from */
#define GLUE_HIGH_END 4 /* other alignment, and to intersect no more
 * than length GLUE_HIGH_END */

#endif
```

parameters.h



```

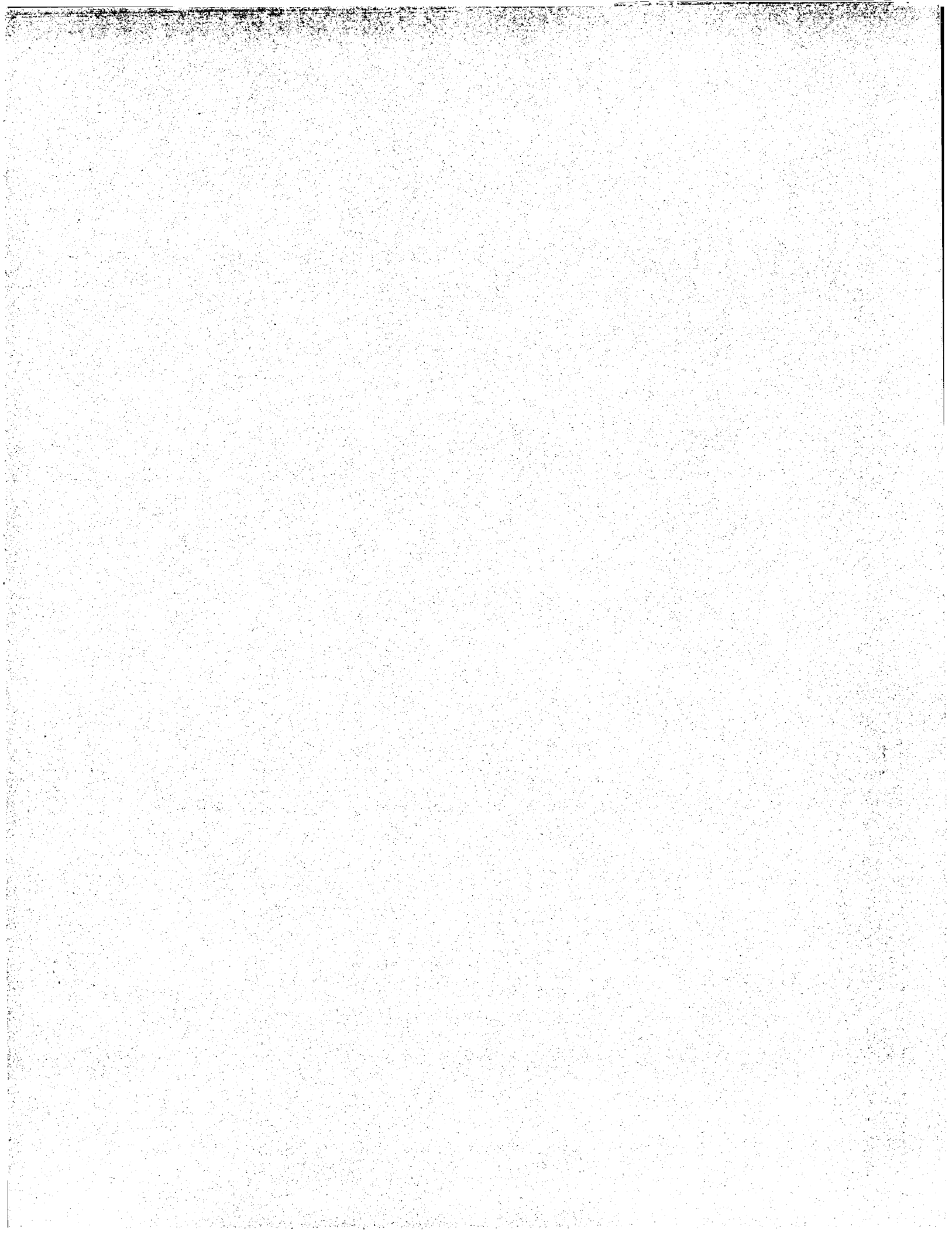
/*
 * $Id: parse.h,v 1.4 1998/02/17 16:16:11 ariels Exp $
 * $Log: parse.h,v $
 * Revision 1.4 1998/02/17 16:16:11 ariels
 * Fix name of global variable cluster_no (to contig_name), since it was
 * neither CLUSTER nor NO (number)....
 *
 * Revision 1.3 1997/11/30 08:15:28 ariels
 * Fixed dud control characters (whoops).
 *
 * Revision 1.2 1997/11/30 08:14:37 ariels
 * Added id and ChangeLog comments.
 *
 */

#ifndef _PARSE_H_
#define _PARSE_H_
#include "hilltops.h"
#include <stdio.h>

int decide (double id_percent, int len, int model);
int check_overlap (char *seq1, char *seq2, hilltop *ht);
void parse_hilltop_information (FILE *hilltops_info_file, char *contig_name);
void get_hilltop_information (char *contig_name);
void invert_sequences ();
int parse(FILE *infile);
void assign_graph_edge(int idx1,int idx2,int st1,int st2,int endl,int end2);
int near_left(int point1,int point2,int idx,int threshold);
int near_right(int point1,int point2,int idx,int threshold);

#endif

```

```

/* $Id: prm.h,v 1.2 1997/11/30 08:17:49 ariels Rel $
 * Revision 1.2 1997/11/30 08:17:49 ariels
 * Fixed dud id string (it was in a static char string, and this is a
 * header file!) and moved it to comment.
 *
 * Revision 1.1 1997/11/30 06:52:48 ariels
 * Initial revision
 *
 * Revision 1.3 92/07/23 16:55:30 gidi
 * *** empty log message ***
 *
 * Revision 1.2 91/11/15 14:57:12 gidi
 * new prm_argv header file
 */

/*
 * define parameter types
 */
#ifndef prm_h
#define prm_h

#include <stdio.h>
#endif

#define ILLEGAL_LINE this is an illegal C line

/* flags for prm_argv mode argument */
#define P_HELP 1
#define P_PRINT 2
#define P_ERROR 4
#define P_IGNORE 8
#define EOLIST 0
#define FALSE 0
#define TRUE 1

/*****
 *
 * parameters */
/*****

/*
 * parameter types :
 * 1. index - get value from a certain argument number
 * 2. field - get value from a "field=value" argument
 * 3. flag - get a boolean value from "flag" argument
 * 4. free - get value from a free argument
 */

enum _prm_id {PR_INDEX, PR_FIELD, PR_FLAG, PR_FREE, PR_MAX_PRM};

/* parameter type structure */

```

```

typedef struct _prm {
    char
    name(20);
    /* parameter type name */
    int
    (*ident)();
    /* identification function */
    int
    (*print)();
    /* print result function */
    int
    (*parse)();
    /* parse function */
    int
    (*get)();
    /* get parameter values function */
    int
    (*help)();
    /* print help function */
    int
    (*destruct)();
    /* destruct instance function */
} prm_type;

/*
 * get function arguments
 * 1. free - return number of free elements
 * 2. arg - get value from an argument instance
 * 3. extern - get value from external source
 * 4. default - get value from parameter default
 */

enum _get_type {D_FREE, D_ARG, D_EXTERN, D_DEFAULT};

int
p_index_ident();
int
p_index_print();
int
p_index_parse();
int
p_index_get();
int
p_index_help();
int
p_index_destruct();

int
p_field_ident();
int
p_field_print();
int
p_field_parse();
int
p_field_get();
int
p_field_help();
int
p_field_destruct();

int
p_flag_ident();
int
p_flag_print();
int
p_flag_parse();
int
p_flag_get();
int
p_flag_help();
int
p_flag_destruct();

int
p_free_ident();
int
p_free_print();
int
p_free_parse();
int
p_free_get();
int
p_free_help();
int
p_free_destruct();

typedef struct _p_index {
    int
    arg_index;
    char
    *format;
} p_index;

typedef struct _p_field {
    char
    *name;
    char
    *def;
    char
    *format;
    int
    maxelems;
    int
    elemsize;
}

```

```

    int      *n_addr;

typedef struct p_flag {
    char      *name;
    int      def;
} p_flag;

typedef struct p_free {
    char      *def;
    char      *format;
} p_free;

/* parameter instance */

typedef struct prm_inst {
    int      type;
    char      *comment;
    void      *addr;
    int      nelems;
    int      external;
    union {
        p_index
        p_field
        p_flag
        p_free
    } p;
    struct prm_inst *next, /* next instance in linked list */
            *prev; /* previous instance in linked list */
} prm_inst;

typedef struct _arg_inst {
    int      used;
    int      count;
    char      *value;
    struct _arg_inst *next, *prev;
} arg_inst;

#ifdef _prm_C

FILE *PrmFp = stderr;
int PrmStringSize = 80;
char *(*PrmExternFunc)() = NULL;

prm_type prm_types[] = {
    ("p_index", p_index_ident, p_index_print, p_index_parse,
     p_index_get, p_index_help, p_index_destruct),
    ("p_field", p_field_ident, p_field_print, p_field_parse,
     p_field_get, p_field_help, p_field_destruct),
    ("p_flag", p_flag_ident, p_flag_print, p_flag_parse,
     p_flag_get, p_flag_help, p_flag_destruct),
    ("p_free", p_free_ident, p_free_print, p_free_parse,
     p_free_get, p_free_help, p_free_destruct),
    ("", NULL, NULL, NULL, NULL, NULL, NULL)
};

char *MatchStr;
#else
extern FILE *PrmFp;

```

prm.h

```

extern int PrmStringSize;
extern prm_type prm_types[];
extern char *MatchStr;
extern char *(*PrmExternFunc)();
#endif

#endif

```

prm.h

```

/* definition of sequence profile */
/*
 * $Id: profile.h,v 1.2 1997/11/30 08:18:23 ariels Rel $
 * $Log: profile.h,v $
 * Revision 1.2 1997/11/30 08:18:23 ariels
 * Added Id and ChangeLog comments.
 */

#ifndef __SEQ_PROF__
#define __SEQ_PROF__

typedef struct seq_prof_node {
    float a,c,g,t,gap,sum;
    struct seq_prof_node *next;
} seq_prof_node;

typedef struct seq_prof {
    int len;
    seq_prof_node *first,*last;
} seq_prof;

seq_prof_node *
seq_prof_node_create(char);
seq_prof_node *
seq_prof_node_free(seq_prof_node *spn);
void
seq_prof_node_update(seq_prof_node *spn, char);
char
seq_prof_node_char(seq_prof_node *spn);

seq_prof *
seq_prof_new();
seq_prof *
seq_prof_create(char *seq);
void
seq_prof_free(seq_prof *sp);
int
prepend_node(seq_prof *sp, seq_prof_node *spn);
int
append_node(seq_prof *sp, seq_prof_node *spn);
int
insert_node(seq_prof *sp, seq_prof_node *prev, seq_prof_node
*new);
seq_prof_node *
get_node(seq_prof *, int ix);
void
seq_prof_update(seq_prof *sp, int start, char *ch);
seq_prof *
seq_prof_concat(seq_prof *sp1, seq_prof *sp2);
seq_prof *
seq_prof_split(seq_prof *sp, int offset);
char *
compute_assembly(seq_prof *, int flag);

char *final_compute_assembly (seq_prof *sp, int ignore_gaps,
int *probable_mistakes, int *corrected);
void
print_seq_prof_node(seq_prof_node *spn);
void
print_seq_prof(seq_prof *sp);
void
check_profile_consistency(seq_prof *sp);

#endif

```



```

#ifndef REPEATS_H
#define REPEATS_H

/*
 * $Log: repeats.h,v $
 * Revision 1.11 1998/04/18 22:11:53 avner
 *   remove definition of MAX_TRANSC that is made in 'general.h'.
 * Revision 1.10 1998/04/15 10:25:12 eyal
 *   Move macros SMALLER BIGGER EQUAL to splice_graph.h.
 * Revision 1.9 1998/04/05 11:46:36 eyal.
 *   Adding a boolean parameter to find_cycle which tells the function weather to
 *   print the cycles or not.
 * Revision 1.8 1998/03/30 09:21:28 eyal
 *   Change the field array of stack from static allocation to dynamic
 * Revision 1.7 1998/03/24 14:00:39 eyal
 *   Adding structs consensus_graph consensus_node consensus_edge
 * Revision 1.6 1998/03/16 06:24:04 eyal
 *   Changin update_points_list_after_split(...) to recieve graph.
 * Revision 1.5 1998/02/21 22:50:55 avner
 *   Changed 'stack_t' to 'stack_type' as the first caused confusion with lib
 *   definition.
 *   Added macros STRAIGHT_REPEAT, INVERTED_REPEAT.
 * Revision 1.4 1998/02/19 14:25:40 eyal
 *   Adding rp_mode_build_transcripts functions
 * Revision 1.3 1998/02/16 14:54:25 eyal
 *   Adding struct stack_type (in use in build rp_mode_transcript)
 * Revision 1.2 1998/02/11 13:42:49 avner
 *   Added Id and ChangeLog comments.
 */

#include "string.h"
#include "splice_graph.h"

#define MAX_PATH_LEN 100

#define STRAIGHT_REPEAT 1
#define INVERTED_REPEAT 2

typedef struct stack_type {
    int *array;
    int size;
    int allocated_size;
    } stack_type;

typedef struct splice_node_container_s {
    splice_node *node;
    struct splice_node_container_s *next;
    } splice_node_container;

typedef struct consensus_edge {

```

repeats.h

```

struct consensus_node *target;
int width; /* The number of transcripts defining this edge */
} consensus_edge;

typedef struct consensus_node {
    int id;
    int replica_id;
    int num_replicas; /* you can rely on this field only in the first replica!!! */
    splice_node *node;
    int out_degree;
    consensus_edge out_neighbors[MAX_NODES];
    } consensus_node;

typedef struct consensus_graph {
    int size;
    consensus_node *node_list[MAX_NODES];
    struct consensus_graph *next;
    } consensus_graph;

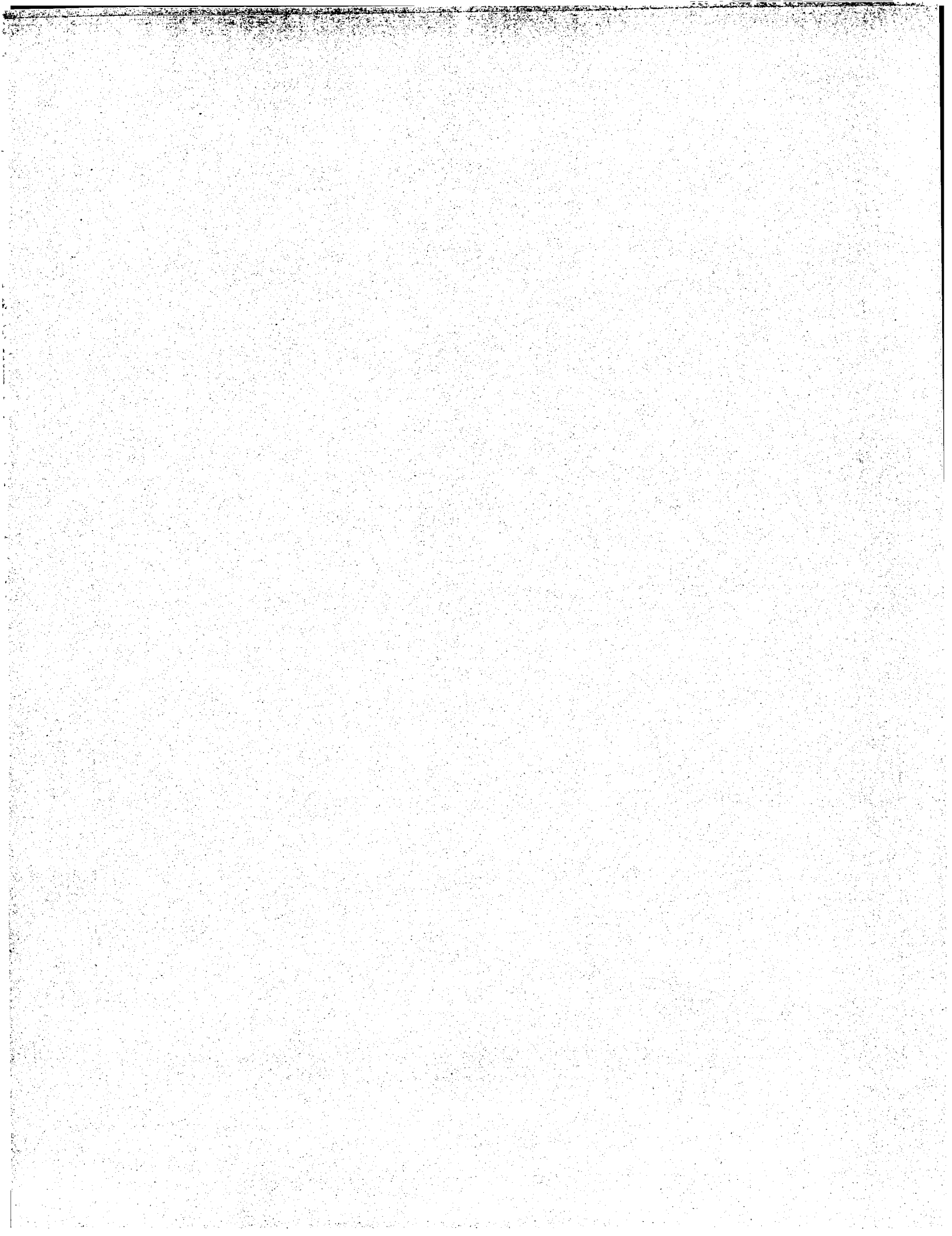
int check_repeat(char *,char *);
void update_points_list_after_unify(graph_point *points_to_update,int left,int right,
    unsigned int left_len,int replaced_node);
void update_points_list_after_split(graph_point * points_to_update,int old_node,
    int left,int right,int offset,splice_graph *
    graph);

splice_graph *rpnode_variant_graph_to_splice_graph(splice_graph *variant_graph);
void check_multiple_hits_in_variants(splice_graph *);
int find_cycle(splice_graph *graph,int print_cycles);
void transcribe_paths_to_transc(splice_graph *graph,
    char *transcripts[MAX_TRANSC],
    int *transc_no, int **paths);
void build_transcripts_paths(splice_graph *graph,int **paths,int *transc_num);
void check_repeats_within_variants(splice_graph *graph);

#endif

```

repeats.h



```

/*
 * Low-level I/O for FastA and rich-FastA format files
 */
/*
 * $Log: rf.h,v $
 * Revision 1.10 1998/04/26 05:47:49 avner
 * remove 'documentation' field in rich_fasta_seq struct.
 *
 * Revision 1.9 1998/04/25 16:49:42 avner
 * add polyt_detected field in struct rich_fasta.
 *
 * Revision 1.8 1998/04/24 12:47:47 avner
 * changed prototype for function 'rich_fasta_read_seq'.
 *
 * Revision 1.7 1998/04/24 09:22:47 avner
 * increase MAX_LINE_LEN and add MAX_HEADER_LINE (both 10000)
 * to deal with really big header lines.
 *
 * Revision 1.6 1998/04/23 17:45:21 avner
 * adding documentation.
 *
 * Revision 1.5 1998/04/19 05:26:34 avner
 * changes due to the different way in which we deal with the 'dirty' fields.
 *
 * Revision 1.4 1998/04/06 08:35:54 avner
 * add field 'polya'.
 *
 * Revision 1.3 1998/03/17 07:35:22 avner
 * changes in fields names for holding both cleaned and original sequences.
 *
 * Revision 1.2 1998/02/22 17:10:08 ariels
 * Read 3 numbers from "#Sz" field.
 *
 * Revision 1.1 1998/02/08 13:33:08 ariels
 * Initial revision
 */

```

```

#ifndef RF_H
#define RF_H

#include <stdio.h>

```

```

#define MAX_LINE_LEN 10000
#define MAX_HEADER_LINE 10000
#define WORD_LEN 20
#define DESC_LEN 100
#define MAX_SEQ_LEN 10000
#define OUTPUT_LINE_WIDTH 60

```

```

typedef struct {
    FILE *fp;
    char last_line[MAX_HEADER_LINE];
    fasta_file_obj *fasta_file;
}
typedef struct region_list {
    int begin,end;
    struct region_list *next;
}

```

rf.h

```

) region_list, *region_list_ptr;

typedef struct rich_fasta {
    char *header;
    char *original_seq;
    char *cleaned_seq;
    char *name;
    char *accession;
    char *id;
    char *clone;
    char *tissue;
    char *library;
    char *chromosome;
    char *definition;
    char *organism;
    int cleaned_len;
    int original_len;
    int clone_len;
    /* location fields */
    int first_clean;
    int first_dirty;
    int quality_mark;
    int polyA;
    int polyT;
    int polyN;
    /****** */
    int polyA_detected; /* flag */
    int polyT_detected; /* flag */
    int trim5; /* location of first char we use in the gb sequence */
    int trim3; /* location of last char we use in the gb sequence */
    /****** */
    int cl_id;
    char *cluster;
    char *meta_cluster;
    int strand; /* +/- 1 */
    char *type;
    int direction; /* 5/3/0 */
    int size[3];
    region_list_ptr vectors;
    region_list_ptr repeats;
    rich_fasta_seq *rich_fasta_seq_ptr;
}

```

```

/* Prototypes */
rich_fasta_seq_new(void);
rich_fasta_seq_free(rich_fasta_seq_ptr rfp);
void

```

```

fasta_file
void
rich_fasta_open(char *file_name);
rich_fasta_close(fasta_file ff);

```

```

/* Full Featured I/O */
rich_fasta_read_seq(fasta_file ff,int application);
rich_fasta_seq_ptr
/* rich fasta for dummies - data is not saved */
int read_next_rich_fasta_seq(fasta_file ff);
char * rich_fasta_seq(void);
char * rich_fasta_name(void);
char * rich_fasta_accession(void);

```

rf.h

Listing for Adam Sartel Sun Aug 9 10:33:29 1998

```
char * rich_fasta__id(void);
char * rich_fasta__clone(void);
int rich_fasta__len(void);
/* more should be here */

/* regular fasta i/o */
int fasta_read_seq(fasta_file ff, char *name, char *doc,
                  int *len, char *data);

void rich_fasta_write(FILE *fp, rich_fasta_seq_ptr rfseq);

void write_sequence(FILE *fp, char *s);

#endif /* ! defined(_RF_H) */
```

```

/*
 * $Id: splice_graph.h,v 1.15 1998/06/29 12:12:51 eval Exp $
 * $Log: splice_graph.h,v $
 * Revision 1.15 1998/06/29 12:12:51 eval
 * Include correct include file.
 *
 * Revision 1.14 1998/06/29 11:30:50 avner
 * add prototypes for cut_est_head and cut_est_tail.
 *
 * Revision 1.13 1998/06/24 07:57:37 eval
 * Add the link parameter to connect_stitch and add_edge.
 *
 * Revision 1.12 1998/06/24 07:22:06 eval
 * 1. Add struct path_node.
 * 2. Add prototype for cprof_align_to_align_data.
 *
 * Revision 1.11 1998/06/18 14:42:47 ariels
 * Strip "#ifdef CPROF_ALIGN" and "#ifdef IMPROVE_CPROF" lines.
 *
 * Revision 1.10 1998/05/11 14:39:59 eval
 * Add functions graph__print_cprofs and splice_node__print_cprof
 *
 * Revision 1.9 1998/04/15 10:21:02 eval
 * add functions:
 * splice_graph__is_initial_node
 * splice_graph__is_ending_node
 * find_nodes_types
 *
 * Revision 1.8 1998/02/23 11:12:26 ariels
 * Fix spelling of "consensus".
 *
 * Revision 1.7 1998/02/21 23:28:25 avner
 * changed prototype as function name changed: was 'are_neighbors',
 * now 'splice_graph__is_directed_edge'.
 *
 * Revision 1.6 1998/02/16 06:16:10 eval
 * Adding field type to splice_node it can be INITIAL or ENDING or NON.
 * the field can be both INITIAL and ENDING so we should look at this field
 * with bit operations.
 *
 * Revision 1.5 1998/01/08 13:45:00 ariels
 * Remove prototypes which don't belong.
 *
 * Revision 1.4 1998/01/07 09:28:55 avner
 * Changing declaration of 'connect_stitch'.
 *
 * Revision 1.3 1997/12/29 16:05:05 eval
 * Changing unify_node and delete_node to return the index of the changed node.
 *
 * Revision 1.2 1997/11/30 08:19:18 ariels
 * Added Id and ChangeLog comments.
 *
 */

#ifndef SPICE_GRAPH_H
#define SPICE_GRAPH_H

/* definition of data types for splice graph */
#include <stdio.h>
#include "dp.h"

```

```

#include "general.h"
#include "profile.h"
#include "cprof.h"
#include "cpof_align.h"
#include "align_data.h"

#define MAX_NODES 200

#define RIGHT_EXTENSION 1
#define LEFT_EXTENSION -1
#define START_POINT 0
#define END_POINT 1

#define NON 0
#define INITIAL 1
#define ENDING 2

#define SMALLER 0
#define BIGGER 1
#define EQUAL 2

typedef struct path_node {
    int node_id;
    int start;
    int end;
    struct path_node *next;
} path_node;

typedef struct graph_point {
    int node;
    int offset;
    int type;
    struct graph_point *next;
} graph_point;

typedef struct no_repeat_seq {
    cprof profile;
    struct no_repeat_seq *next;
} no_repeat_seq;

typedef struct est_list_s {
    int est;
    struct est_list_s *next;
} *est_list;

/*****
typedef struct arc {
    int idx;
    int width;
    int is_ximeric;
    est_list ests_passing;
} arc;

typedef struct splice_node {
    int id;
    char *seq;
    cprof profile;
    arc out_neighbors[MAX_NODES];
    arc in_neighbors[MAX_NODES];

```

```

int in_degree, out_degree;
int marked;
int variant_depth;
int need_to_update;
int start_update_into;
int start_update_with;
char *align_str;
align_data *align_list;
int err_num;
int corrected_err;
int type; /* INITIAL | ENDING | NON */
} splice_node;

splice_node * node_new(void); /* create an empty node */
void node_strset(splice_node *node, char *str, int from, int to);
/* copy string (in specified region) to a node */
void node_setid(splice_node *, int id); /* set id */

/*****
typedef struct splice_graph {
    int size;
    splice_node *node_list[MAX_NODES];
} splice_graph;

splice_graph * splice_graph_new(void); /* create an empty graph */
int add_node(splice_graph *, splice_node *);
void add_edge(splice_graph *, int node1, int node2, int link);
int splice_graph_is_directed_edge(splice_graph *,
    int node1, int node2);
int splice_graph_is_undirected_edge(splice_graph *,
    int node1, int node2);
void graph_free(splice_graph *);
splice_graph __align_lists_free(splice_graph *graph, int best_var
    iant);
int splice_graph_in_deg(splice_graph *, int);
int splice_graph_out_deg(splice_graph *, int);
int split_node(splice_graph *, int, int, int *, int *);
char *get_alignment_string(hilltop *ht, char *second_seq);

splice_node * node_init(cprof cprof);
void update_node(splice_node *, cprof);
void fix_update_node_parameters(splice_node *node,
    int st_offset_into, int st_offset_with,
    char *str);
align_data *align_node_to_cprof(splice_node *, cprof); /* compute alignment */

void graph_print(splice_graph *graph);
void splice_node_print(splice_node *node, int id);

void unify_needed_nodes(splice_graph *graph);
int unify_nodes(splice_graph *graph, int node_left, int node_right);
void replace_occurrences_in_neighbors_list(splice_graph *graph, int old_idx, int ne
    w_idx);
int graph_delete_node(splice_graph *graph, int node_idx, int profile_mem_free);
void splice_graph_increment_variant_depth(splice_graph *graph, int idx);

int connect_stitch(splice_graph *graph, graph_point *point1, graph_point *point2

```

```

int splice_graph __is_initial_node(splice_graph *graph, int node);
int splice_graph __is_ending_node(splice_graph *graph, int node);
int find_nodes_types(splice_graph *graph);
void graph_print_cprofs(splice_graph *graph, FILE *fp);
void splice_node_print_cprof(splice_node *node, int id, FILE *fp);
align_data *cprof_align_to_align_data(cprof_align cpal, cprof pl, cprof p2,
    int node_id);
void cut_est_head(int est_idx, int est_head_cupoint, splice_graph *graph);
void cut_est_tail(int est_idx, int est_tail_len, splice_graph *graph);
/*****
int *splice_graph_get_matrix(splice_graph *);
#endif

```

```
/* Klugy-kode-include file!      Emacs: -*- c -*- */
```

```
/*
 * There is a special place in programming hell reserved for people
 * who write this sort of thing. On the other hand, it's a neat
 * solution for when you need to perform the same sort of update 3
 * times, but need a slightly different update each time.
 */
```

```
/* Before #including this file, #define the DRAG_ARGS macro.
 * DRAG_ARGS(this, that, val) should update this cell of the matrix
 * from that cell, on the assumption that this cell should get the
 * score val.
 */
```

```
/* The legal state transitions are:
 * e_ll -> e_ll (3 ways) - by paying likelihood costs.
 * e_ll -> e_lgap1 or e_lgap2 - by paying lgap_open.
 * e_lgap1 -> e_lgap1 - by paying lgap_extend (0).
 * e_lgap2 -> e_lgap2 - by paying lgap_extend (0).
 * e_lgap1 -> e_lgap2 - by paying lgap_extend (0).
 */
```

```
/* Note that when lgap_extend is 0, having the last transition is
 * exactly like having the ALT state of the six18 model: there is no
 * advantage in moving diagonally (as in the ALT state) over moving
 * horizontally and vertically (as in combinations of the YGAP and
 * XGAP states) when steps cost nothing.
 */
```

```
/*
 * $Log: ac_update.k,v $
 * Revision 1.6 1998/02/04 16:38:43 ariels
 * Fix bug which caused incorrect score calculation in squish_lgaps().
 * YOU MUST USE THIS WITH REVISION 1.16 OF ALIGN_CPROF.C!
 */
```

```
/* Revision 1.5 1998/01/08 13:22:44 ariels
 * Make parameter block (prm) static in align_cprof.c, rather than an
 * externally supplied set of values.
 */
```

```
/* Revision 1.4 1998/01/05 10:13:54 ariels
 * Explain state transitions in header comment.
 */
```

```
/* * Fix comment leader (both in RCS repository and in old comments).
```

```
/* Revision 1.3 1997/12/31 10:39:38 ariels
 * Removed 'cost' field from profiles; instead of it are a pair of static
 * variables in align_cprof.c which hold the location scores for the pair
 * of profiles being aligned.
 */
```

```
/* Revision 1.2 1997/12/17 10:23:12 ariels
 * (cosmetics)
 */
```

```
#define MAXIMISE(this,that,val) \
    if ((val) > (this).sc)    DRAG_ARGS(this, that, val)
```

```
{
    cprof_node N;
    int k;
    double score;
```

ac_update.k

```
/* ----- Update match scores ----- */
```

```
/* Score to match */
for(k=0; k<N_PROFILE_LETTERS; k++){/* proposed profile */
    N[k] = pl->node[i][k] + p2->node[j][k];
    score = (*diag)[e_ll].sc + ll_func(N) - cost1[i] - cost2[j];
    DRAG_ARGS((*current)[e_ll], (*diag)[e_ll], score);
}
```

```
/* Score to open a gap in pl */
for(k=0; k<N_PROFILE_LETTERS; k++){
    N[k] = p2->node[j][k];
    N[e_gap] += pl->link[i];
    score = (*down)[e_ll].sc + ll_func(N) - cost2[j];
    MAXIMISE((*current)[e_ll], (*down)[e_ll], score);
}
```

```
/* Score to open a gap in p2 */
for(k=0; k<N_PROFILE_LETTERS; k++){
    N[k] = pl->node[i][k];
    N[e_gap] += p2->link[j];
    score = (*left)[e_ll].sc + ll_func(N) - cost1[i];
    MAXIMISE((*current)[e_ll], (*left)[e_ll], score);
}
```

```
/* Score to close an X-gap */
score = (*down)[e_lgap1].sc + lgap_open(pl->link[i], p2->link[j]);
MAXIMISE((*current)[e_ll], (*down)[e_lgap1], score);
```

```
/* Score to close a Y-gap */
score = (*left)[e_lgap2].sc + lgap_open(p2->link[j], pl->link[i]);
MAXIMISE((*current)[e_ll], (*left)[e_lgap2], score);
```

```
/* ----- Update X gap scores ----- */
```

```
/* score to extend a long X gap */
score = (*down)[e_lgap1].sc +
    lgap_extend(pl->link[i], p2->link[j]);
DRAG_ARGS((*current)[e_lgap1], (*down)[e_lgap1], score);
```

```
/* score to extend from a long Y gap */
/* score = (*left)[e_lgap2].sc + lgap_extend(p2->link[j], pl->link[i]);
MAXIMISE((*current)[e_lgap1], (*down)[e_lgap2], score);*/
```

```
/* score to open an X-gap */
score = (*current)[e_ll].sc + lgap_open(pl->link[i], p2->link[j]);
MAXIMISE((*current)[e_lgap1], (*current)[e_ll], score);
```

```
/* ----- Update Y gap scores ----- */
```

```
/* Score to extend a long Y gap */
score = (*left)[e_lgap2].sc +
    lgap_extend(p2->link[j], pl->link[i]);
DRAG_ARGS((*current)[e_lgap2], (*left)[e_lgap2], score);
```

```
/* Score to extend from a long X gap */
score = (*current)[e_lgap1].sc + lgap_extend(pl->link[i], p2->link[j]);
MAXIMISE((*current)[e_lgap2], (*left)[e_lgap1], score);
```

```
/* Score to open a Y-gap */
score = (*current)[e_ll].sc + lgap_open(p2->link[j], pl->link[i]);
```

ac_update.k

```
MAXIMIZE(*current)[e_lgap2], (*current)[e_ll], score);
```

```
)
```

```
#undef MAXIMIZE
```

Listing for Adam Sartiell

Sun Aug 19 10:38:38 1998

```

# Makefile for assembly
#
# Platform independent! Make sure your .cshrc (or equivalent) sets
# the environment variable ARCH to one of 'DEC', 'SUN', 'SGI', 'LINUX'
# or things won't work.
#
# Now type "make CFLAGS=-g" to debug, "make CFLAGS=-O" to optimise,
# "make COMP=gcc" to force compilation with the GNU compiler, etc.,
# etc.
#
# comments to ariels@compugen, please.
#
# $Log: Makefile,v $
# Revision 1.22 1998/07/05 07:42:40 ariels
# Remove profile.[ch], AGAIN. It crept back into the previous version!
#
# Revision 1.21 1998/07/02 11:32:43 eyal
# make purify mode create a .pure file
#
# Revision 1.19 1998/06/25 09:04:16 ariels
# Add "mem.c" src file
#
# Revision 1.18 1998/05/21 12:41:15 ariels
# Better (dummy) purify target, which works with new-style purify on Sun
# (requires the entire link run) and allows PURIFY=quantify to run quantify
# instead.
#
# Revision 1.17 1998/04/28 09:44:39 ariels
# Added sort.c to sources for assembly.DEC
#
# Revision 1.16 1998/04/26 13:55:57 ariels
# Accept MCOMP= variable (on command-line, too) which (if specified) is
# used for making dependencies. $(MCOMP) should be a C compiler which
# accepts the '-M' flag (e.g. gcc). This is useful for compiling on a
# platform where your main compiler doesn't accept the flag (e.g. Sun).
#
# Revision 1.15 1998/04/02 06:19:39 ariels
# Check MAKECMDGOALS for the goal 'clean'. If we're running "make
# clean", avoid remaking (and re-including) the dependencies -- we're
# going to delete them anyway in a moment!
#
# Revision 1.14 1998/02/19 14:32:33 eyal
# Adding files hyper_graph.[ch]
#
# Revision 1.13 1998/02/17 16:21:09 ariels
# Correct 'purify' target.
#
# Revision 1.12 1998/02/11 16:25:09 ariels
# Support separate full-cluster flipper program.
#
# Revision 1.11 1998/02/11 10:16:53 avner
# Added repeats.[ch] and analyze_graph.[ch] (formed from splitting build_c).
#
# Revision 1.10 1998/02/08 13:27:15 ariels
# Separated rich-FastA I/O routines into rf.[ch]
#
# Revision 1.9 1998/02/01 14:46:33 ariels
# Added "assembly.$(ARCH).pure" target to run purify.
#
# Revision 1.8 1998/01/08 13:21:49 ariels

```

Makefile

Listing for Adam Sartiell

Sun Aug 9 10:38:38 1998

```

# Better handling of existing object-file directories.
#
# Revision 1.7 1997/12/24 08:07:10 ariels
# Fixed stupid bug which created a file called '1' whenever the .d files
# were remade (don't ask).
#
# Revision 1.6 1997/12/17 11:59:49 ariels
# Added id.c (static compilation identifiers) to SRCS
#
# Revision 1.5 1997/12/14 10:13:05 ariels
# Now uses GNU make to simplify automatic dependency generation, have
# cleaner command lines, etc.
#
# Note that GNU make *must* be used!
#
# Revision 1.4 1997/12/03 10:36:09 ariels
# Added 'TAGS' target (including .h files)
#
# Revision 1.3 1997/11/30 08:50:53 ariels
# Made mfile_ver work
#
# Revision 1.2 1997/11/30 08:22:22 ariels
# Added Changelog comment and "mfile_ver" pseudo-target to get the Id
# string.
#
MFILE_VER = $$Id: Makefile,v 1.22 1998/07/05 07:42:40 ariels Exp $$
CC = $(COMP)
ifdef MCOMP
CCM = $(MCOMP)
else
CCM = $(COMP)
endif
ifdef PURIFY
PURIFY = purify
endif
OBJDIR = $(ARCH)obj
CFLAGS = -I/home/raveh/include $(CCFLAGS)
CL = $(COMP) -L$(RAVEHLIBDIR)
LIBS = -lprn -lm
# for debugging...
#LIBS = libprn/libprn.a -lm

all: assembly.$(ARCH) flipper.$(ARCH)

mfile_ver:
@echo \$(MFILE_VER)

ASS_SRCS=main.c io.c build_graph.c rules.c splice_graph.c olap_graph.c \
custody_zones.c est_table.c output.c error.c dp.c \
hilltops.c align_cprof.c cprof.c cprof_align.c id.c \
rf.c analyze_graph.c repeats.c hyper_graph.c sort.c mem.c

FLIPPER_SRCS=flipper.c rf.c parse.c est_table.c error.c dp.c hilltops.c \
output.c

ASS_INCS=align_data.h custody_zones.h io.h align_prm.h \
dp.h olap_graph.h splice_graph.h build_graph.h

```

Makefile

A-206

```
est_table.h parameters.h cprof.h general.h parse.h \
cprof_align.h hilltops.h prm.h rf.h analyze_graph.h repeats.h \
hyper_graph.h

FLIPPER_INCS=rf.h parse.h est_table.h olap_graph.h error.h general.h

SRCS=$(ASS_SRCS) $(FLIPPER_SRCS)
INCS=$(ASS_INCS) $(FLIPPER_INCS)

TAGS:  $(SRCS) $(INCS)
        etags -t $(SRCS) $(INCS)

# Dependencies
ifneq ($(MAKECMDGOALS),clean)
    include $(SRCS:%.c=$(OBJDIR)/%.d)
endif

$(OBJDIR)/%.d: %.c
    @$(SHELL) -ec '[ -d $(OBJDIR) ] || mkdir $(OBJDIR)'
    $(SHELL) -ec '$(CC) -M $(CPPFLAGS) $< \
    | sed -e '\''s|\\$(*)|\\.o|:|$(OBJDIR)/1.o $@ : |g\\'' \
    -e '\\s|usr/include|[^ ]*|g\\'' > $@; \
    [ -s $@ ] || rm -f $@'

endif

ASS_OBJS=$(ASS_SRCS:%.c=$(OBJDIR)/%.o)
FLIPPER_OBJS=$(FLIPPER_SRCS:%.c=$(OBJDIR)/%.o)

assembly $(ARCH): $(ASS_OBJS)
    $(CL) $(CFLAGS) -o assembly.$(ARCH) $(ASS_OBJS) $(LIBS)

flipper.$(ARCH): $(FLIPPER_OBJS)
    $(CL) $(CFLAGS) -o flipper.$(ARCH) $(FLIPPER_OBJS) $(LIBS)

#assembly.$(ARCH).purify: $(OBJECTS)
#    purify $(CL) $(CFLAGS) -o assembly.$(ARCH) $(OBJECTS) $(LIBS)
assembly.$(ARCH).pure: assembly.$(ARCH)
    ${PURIFY} $(CL) $(CFLAGS) -o assembly.$(ARCH) $(ASS_OBJS) $(LIBS)

$(OBJDIR)/%.o:
    $(CC) $(CPPFLAGS) $(CFLAGS) -c $< -o $@

.PHONY: clean
clean:
    rm -f $(OBJDIR)/*.o $(OBJDIR)/*.d assembly.$(ARCH) core
```

```
static char rcsid[] = "$Id: cln.c,v 1.8 1998/06/01 11:21:03 prod Exp $";
```

```
#include <stdio.h>
#include <malloc.h>
#include <math.h>
#include <pm.h>
#include "hilltops.h"
#include "wdp.h"

#define MIN_LEN 10
#define LINE_SIZE 5000
#define MAX_SEQ_LEN 50000
#define MIN_REP_LEN 20

/* function prototypes */
int get_bic_info (FILE *bf, char *qname, float *score, int *start, int *end);
void add_to_list (hilltop **list, hilltop ht);
int compare_hilltops (hilltop *ht1, hilltop *ht2);
int fasta_read (FILE *fp, char *name, char *comment, char *data, char *line);
int cut_est (char *seq, hilltop *list, char *name, char *comment, char *map);
int n_cut (char *seq);
void parse_comment (char *comment, int len_seq, char *date_st);
void remove_comment_name (char *comment);
int look_for (char *pat, char *st, int start);
char uppercase (char c);

/* Global variables */
FILE *text_file, *xout_file, *unmasked_file, *vf, *rf, *fp;
int mrna, mtrans, no_cut_n, pbound;
int gapext, gapop, match, mismatch, edge_bound, inner_bound, cutoff;

/* ===== */
int main (int argc, char *argv[])
{
    char dir[100], fname[100], vname[100], rname[100], name[100];
    char xout_name[100], unmasked_name[100], text_name[100];
    int v_end_flag, r_end_flag;
    char v_seq_name[30], r_seq_name[30], seq_name[100], comment[LINE_SIZE];
    float v_score, r_score;
    int v_x0, v_xt, r_x0, r_xt;
    int seqno, rc, i, len, len2;
    hilltop *list;
    char seq[MAX_SEQ_LEN];
    float sum, sumsq, avg, var, score;
    int thrown_seqs, thrown_bases, untouched_seqs, untouched_bases;
    int cut_bases;
    int print_version;

    pmc_argv (argc, argv, P_PRINT | P_HELP,
        "dir = %s : directory", dir,
        "in = %s : input file name", fname,
        "vec.in = .vbic %s : vector bicfile", vname,
        "rep.in = .rbic %s : repeat bicfile", rname,
        "mrna = %s : type of data is rna", mrna,
        "mtrans = %s : type of data is transcripts", mtrans,
        "pbound = 220 %d : bound for documenting vectors/repeats", &pbound,
```

cln.c

```
"gapext = 22 %d : multiplicative constant for gaps", &gapext,
"gapop = 33 %d : additive constant for gaps", &gapop,
"match = 10 %d : score for match", &match,
"mismatch = -48 %d : penalty for mismatch", &mismatch,
"edge_bound = 60 %d : bound for edge l.c. regions", &edge_bound,
"inner_bound = 100 %d : bound for inner l.c. regions", &inner_bound,
"cutoff = 60 %d : cutoff for hilltops", &cutoff,
"-ver %s : print version details", &print_version,
"-no_cut_n %d : dont cut dirty tails on Ns", &no_cut_n,
EOLIST);

if (print_version) {
    fprintf(stderr, "RCS Version information: \"%s\\n\", rcsid);
    exit(1);
}

if (mrna && mtrans) {
    fprintf(stderr, "You can't use both -mrna and -mtrans\\n");
    exit(1);
}

init_matrix (match, mismatch, gapext, gapop);

/* Open output files */
sprintf (xout_name, "%s/%s.xout", dir, fname);
sprintf (unmasked_name, "%s/%s.unsk", dir, fname);
sprintf (text_name, "%s/%s.info", dir, fname);
if ((text_file = fopen (text_name, "w")) == NULL)
    error (1, -1, "Can't open output file %s\\n", text_name);
if ((xout_file = fopen (xout_name, "w")) == NULL)
    error (1, -1, "Can't open output file %s\\n", xout_name);
if ((unmasked_file = fopen (unmasked_name, "w")) == NULL)
    error (1, -1, "Can't open output file %s\\n", unmasked_name);

/* Open input files */
/* no vectors for transcripts! */
if (! mtrans) {
    sprintf (name, "%s/%s", dir, fname, vname);
    if ((vf = fopen (name, "r")) == NULL)
        error (1, -1, "Can't open vector file %s\\n", name);
    sprintf (name, "%s/%s", dir, fname, rname);
    if ((rf = fopen (name, "r")) == NULL)
        error (1, -1, "Can't open repeats file %s\\n", name);
    sprintf (name, "%s/%s", dir, fname);
    if ((fp = fopen (name, "r")) == NULL)
        error (1, -1, "Can't open file %s\\n", name);
}

/* no vectors for transcripts! */
if (! mtrans) {
    v_end_flag = get_bic_info (vf, v_seq_name, &v_score, &v_x0, &v_xt);
    r_end_flag = get_bic_info (rf, r_seq_name, &r_score, &r_x0, &r_xt);
}

/* Main loop */
for (seqno = rc = 0; rc; seqno++) {
    /* Get next sequence and initialize map */
    rc = fasta_read (fp, seq_name, comment, seq, line);
    if ((map = (char *) malloc (strlen (seq)+1)) == NULL)
```

cln.c


```

error( 1, -1, "Cannot allocate memory for map\n");
for (i = 0; i <= strlen(seq); i++) map[i] = 0;

/* no vectors for transcripts! */
if ( !trans ) {
    /* look for documented vector hits and update */
    for (i; !strcmp(v_seq_name, seq_name) && !v_end_flag;) {
        create_hilltop (&ht);
        ht->x0 = v_x0;
        ht->xt = v_xt;
        ht->score = v_score;
        sprintf (ht->name, v_seq_name);
        ht->type = 0;
        add_to_list (&list, *ht);
        for (i = ht->x0; i <= ht->xt; i++) map[i] = 1;
        destroy_hilltop (&ht);
        if (!v_end_flag)
            v_end_flag = get_bic_info (vf, v_seq_name, &v_score, &v_x0, &v_xt);
    }
}

/* look for documented repeat hits and update */
for (i; !strcmp (r_seq_name, seq_name) && !r_end_flag;) {
    if ( !trans || r_xt - r_x0 + 1 > MIN_REP_LEN ) {
        create_hilltop (&ht);
        ht->x0 = r_x0;
        ht->xt = r_xt;
        ht->score = r_score;
        sprintf (ht->name, r_seq_name);
        ht->type = 1;
        add_to_list (&list, *ht);
        for (i = ht->x0; i <= ht->xt; i++) map[i] = 1;
        destroy_hilltop (&ht);
    }
    if (!r_end_flag)
        r_end_flag = get_bic_info (rf, r_seq_name, &r_score, &r_x0, &r_xt);
}

/* Decide what to cut */
len = strlen (seq);
fprintf (stderr, "%d Working on %s\n", seqno, seq_name);
len2 = cut_est (seq, list, seq_name, comment, map);
if (list) destroy_hilltop_list (&list);
free (map);

/* Take care of statistics */
sum += len;
sumsq += (double)len*(double)len;
if (len2 == 0) thrown_seqs++;
if (len2 == len) {
    untouched_seqs++;
    untouched_bases += len;
}
thrown_bases += len - len2;
if (len2 != 0) cut_bases += len - len2;
}

/* Print statistics */
avg = sum / (double)seqno;
var = sumsq / (double)seqno - avg*avg;

```

c/n.c

```

fprintf (text_file, "\n\nRead %d sequences.\n", seqno);
fprintf (text_file, "Sum of lengths %5.2f. sumsq of lengths %5.2f\n",
        sum, sumsq);
fprintf (text_file, "Average length: %4.2f, Sd: %4.2f\n", avg, sqrt (var));
fprintf (text_file, "Threw away %d sequences altogether.\n", thrown_seqs);
fprintf (text_file, "Threw away %d bases totally.\n", thrown_bases);
fprintf (text_file, "Of these, cut %d bases from surviving sequences.\n",
        cut_bases);
fprintf (text_file, "Left %d sequences untouched.\n", untouched_seqs);
fprintf (text_file, "They include %d bases totally.\n", untouched_bases);
fprintf (text_file, "Done !!!\n");

/* Cleanup */
fclose (text_file);
fclose (xout_file);
fclose (xout_file);
fclose (unmasked_file);
fclose (rf);
fclose (vf);

exit(0);
}

/*=====
int get_bic_info (FILE *bf, char *qname, float *score, int *start, int *end)
/*
* Read the file compiled from bic (or blast) outputs. This file should have
* in each line 6 fields:
* a. repeat or vector name.
* b. query sequence name (the file should be sorted by this field).
* c. Strand of hit (+ or -).
* d. Score of hit.
* e. Start base of hit in query sequence.
* f. End base of hit in query sequence.
*/
(
    int i, n;
    char dname[80], sign[80], st[200], *rc;

    rc = fgets (st, 200, bf);
    n = sscanf (st, "%s %s %s %f %d %d", dname, qname, sign,
                score, start, end);
    if (rc == NULL || n < 6) return (1);
    return (0);
)
/*=====
void add_to_list (hilltop **list, hilltop ht)
{
    hilltop **htpp, *tmp, *tmp2;
    int val;

    create_hilltop (&tmp);
    copy_hilltop (ht, tmp);

    for (htpp = list; *htpp != NULL; htp = &(*htpp->next)) {
        val = compare_hilltops (*htpp, tmp);
        if (val == 1) { /* The new hilltop can be thrown */
            destroy_hilltop (&tmp);
            return;
        } else if (val == 2) { /* Replace existing hilltop */
            tmp->next = (*htpp->next);

```

c/n.c

```

tmp2 = (*http);
*http = tmp;
destroy_hilltop (&tmp2);
}

/* No other hilltop could be compared to ours - add it to the list */
for (http = list; *http != NULL; http = &((*http)->next)) {
    if ((*http)->x0 > tmp->x0) {
        tmp->next = *http;
        *http = tmp;
        return;
    }
}

/* If we got this far - the new hilltop should be added last */
tmp->next = NULL;
*http = tmp;
}

/*-----*/
int compare_hilltops (hilltop *ht1, hilltop *ht2)
{
    if (ht1->x0 <= ht2->x0 && ht1->xt >= ht2->xt) return (1);
    if (ht1->x0 >= ht2->x0 && ht1->xt <= ht2->xt) return (2);
    else return (0);
}

/*-----*/
int fasta_read (FILE *fp, char *name, char *comment, char *data, char *line)
{
    /* Read FastA format. line remembers the last line read, so it should be
     * emptied when starting a new file.
     */
    int rc, i, j;

    /* Read header line (if necessary) */
    if (strlen (line) == 0) {
        for (rc = 0; rc != EOF;) {
            rc = (fgetc (line, LINE_SIZE, fp) == NULL);
            if (line[0] == '>') break;
        }
        if (rc) {
            line[0] = '\0';
            return (rc);
        }
    }

    /* copy comment */
    strcpy (comment, line, LINE_SIZE);
    comment[strlen (comment)-1] = '\0';

    /* Extract name */
    for (i = 0; i++) {
        if (line[i+1] == ' ' || line[i+1] == '\0' || line[i+1] == '\n') break;
        name[i] = line[i+1];
        name[i+1] = '\0';
    }
}

```

cln.c

```

/* Read data */
for (rc = j = 0; !rc;) {
    rc = (fgetc (line, LINE_SIZE, fp) == NULL);
    if (line[0] == '>' || rc) break;
    for (i = 0; i < strlen (line)-1; i++)
        data[i+j] = uppercase (line[i]);
    j += strlen (line)-1;
    data[j] = '\0';
}
if (rc)
    line[0] = '\0';
return (rc);
}

/*-----*/
#define REALLY_BAD 6
#define PERCENTAGE 3
#define FRAME 25
#define MANY_N 3
#define MIN_EST_OUT 80
#define CUT_GAP 10
#define MIN_LOW_LEN 20

int cut_est (char *seq, hilltop *list, char *name, char *comment, char *map)
{
    int i, j, k, l, q, n, bt, len, n_pos, flag, com_len, point;
    int xout_start, xout_end, umsk_start, umsk_end;
    hilltop *p, *low_complexity_list;
    hilltop *full_lc_list = NULL, *vector_list = NULL;
    char st[30], start_st[20], end_st[20], type_st[10];
    poly st[3], NUCL[4] = "ACGT", nl_st[30], date_st[50];
    int poly_a_start, poly_t_end;
    int rep_len;

    len = strlen (seq);

    /* for transcript - no need for Q and BT */
    if (!mtrans) {
        /* first, parse comment for L, Q, and BT */
        com_len = strlen (comment);
        for (i = 2; i < com_len; i++)
            if (comment[i] == 'N' && comment[i-1] == 'L' && comment[i-2] == '#')
                break;

        if (i == com_len) {
            bt = 0; q = 0; l = len;
            /* error (0, 0, "Bad comment line for sequence %s\n", name);
             * return (0); */
        } else {
            n = sscanf (&comment[i+2], "%d %QL %d %BT %d", &l, &q, &bt);
            if (n != 3) {
                printf ("Bad quality-information (%d %d %d %d)\n%s\n", n, l, q, bt, comment);
                exit (1);
            }
        }

        /* Now create new comment */
        parse_comment (comment, strlen (seq), date_st);
    }
}

```

cln.c

A-210

```

    }
    else {
        /* remove the name from the comment - leave only the real comment */
        remove_comment_name(comment);
    }

    /* create text output: first a line with statistics */
    if ( !mtrans ) {
        fprintf (text_file, "%s length = %d\n", name, len);
    }
    else {
        fprintf (text_file, "%s length = %d Q = %d BT = %d", name, len, q, bt);
        if ( !l != len )
            fprintf (text_file, " Wrong length (should be %d).\n", len);
        else fprintf (text_file, "\n");
    }

    if (mrna) {
        fprintf (type_st, "#TY RNA ");
    }
    else if (mtrans) {
        fprintf (type_st, "");
    }
    else {
        fprintf (type_st, "#TY EST ");
    }

    /* Next: information about cleaned regions */
    for (p = list; p != NULL; p = p->next) {
        if (p->type) strcpy (st, "repeat");
        else strcpy (st, "vector");
        fprintf (text_file, "%s %4d %4d %7s %5d\n", name, p->x0,
            p->xt, st, (int)p->score);
    }

    /* If the sequence is too short we exit here */
    if (len < MIN_EST_OUT) {
        fprintf (text_file, "%s too short - deleted\n\n", name);
        return (0);
    }

    /* Don't cut Ns for transcripts */
    if ( !mtrans || no_cut_n ) {
        n_pos = len;
    }
    else {
        /* Get the N's information and display it */
        n_pos = n_cut (seq);
        fprintf (text_file, "%s %4d %4d N-info\n", name, n_pos, len);
    }

    xout_start = 0;

    /* no vectors for transcripts! */
    if ( !mtrans ) {
        /* Decision: first the start. This is easy since list is sorted by x0 */

```

c/n.c

```

    for (p = list; p != NULL; p = p->next) {
        if (p->type) continue; /* known repeat - mask, don't cut */
        point = (p->xt > len) ? len : p->xt;
        if ((p->x0 - xout_start < CUT_GAP) && (xout_start < point)) {
            xout_start = point;
            add_to_list (&vector_list, *p);
        }
    }

    umsk_start = xout_start;
    xout_end = n_pos;
    umsk_end = len;

    /* no vectors for transcripts! */
    if ( !mtrans ) {
        /* Decision: Now the end (harder). We start with the N and external info
        for the xout file, len for umsk file. */

        if (q > 0 && xout_end > q + 20) xout_end = q + 20;

        for (flag = 1; flag;) {
            flag = 0;
            for (p = list; p != NULL; p = p->next) {
                if (p->type) continue; /* known repeat - don't cut */
                point = (p->x0 < 0) ? 0 : p->x0;
                /* check for change in end of xout file */
                if (xout_end - p->xt < CUT_GAP && xout_end > point) {
                    xout_end = point;
                    flag = 1; /* Change has occurred - we have to loop again */
                }
                /* check for change in end of umsk file */
                if (umsk_end - p->xt < CUT_GAP && umsk_end > point) {
                    umsk_end = point;
                    add_to_list (&vector_list, *p);
                    flag = 1; /* Change has occurred - we have to loop again */
                }
            }
        }

        /* Look for low-complexity regions - non transcripts */
        if ( !mtrans ) {
            poly_a_start = len;
            poly_t_end = -1;
            for (i = 0; i < 4; i++) {
                polyst[i] = NUCL[i];
                polyst[i+1] = '\0';
                low_complexity_list = WHillTops (seq, polyst, cutoff, edge_bound);
                for (p = low_complexity_list; p != NULL; p = p->next) {
                    /* add to full low-complexity list (for umsk version output) */
                    add_to_list (&full_lc_list, *p);
                    /* if the low-complexity region is inside the good xout part -
                    check it */
                    if (p->x0 <= xout_end && p->xt >= xout_start) {
                        /* if we survived - look for l.c. regions overlapping start or
                        xout_end */
                        if (p->x0 <= xout_start + MIN_LOW_LEN ||
                            p->xt >= xout_end - MIN_LOW_LEN) {
                            fprintf (text_file, "%s %d %d end low complexity\n", name,

```

c/n.c

```

        p->xt);
    if (p->x0 <= xout_start + MIN_LOW_LEN) {
        xout_start = p->xt;
    }
    else {
        xout_end = p->x0;
    }
    } else {
        /* In the middle */
        if (p->score >= inner_bound) {
            fprintf(text_file, "%s %d %d middle low complexity\n", name,
                    p->x0, p->xt);
            for (j = p->x0; j <= p->xt; j++) {
                map[j] = 1;
            }
        }
        /* if the low-complexity region is inside the good umsk part -
        check for poly-A at end, or poly-T at beginning
        WARNING - we rely upon the fact that the looping of the
        low complexity regions looks only for poly-X, and uses
        the NUCL array */
        if (p->x0 <= umsk_end && p->xt >= umsk_start) {
            if (strcmp(polyst, "A") == 0 && p->xt >= umsk_end - MIN_LOW_LEN && p->x
0 < poly_a_start) {
                poly_a_start = p->x0;
            }
            else if (strcmp(polyst, "T") == 0 && p->x0 <= umsk_start + MIN_LOW_LEN
&& p->xt > poly_t_end) {
                poly_t_end = p->xt;
            }
        }
        destroy_hilltop_list (&low_complexity_list);
    }
    else { /* transcripts */
        poly_a_start = len;
        poly_t_end = -1;
        polyst[2] = '\0';
        for (i = 0; i < 4; i++) {
            polyst[i] = NUCL[i];
            for (j = i; j < 4; j++) {
                polyst[ij] = NUCL[j];
                if (j == i) polyst[ij] = '\0';
            }
            /* Now polyst is something like "A", "T", ....., "AC", "AT", ... */
            low_complexity_list = WHillTops (seq, polyst, cutoff, edge_bound);
            for (p = low_complexity_list; p != NULL; p = p->next) {
                /* if the low-complexity region is inside the good xout part -
                check it */
                if (p->x0 <= xout_end && p->xt >= xout_start) {
                    /* if we survived - look for l.c. regions overlapping start or
                    xout_end */
                    rep_len = p->xt - p->x0 + 1;
                    if (p->x0 <= xout_start + MIN_LOW_LEN ||
                        p->xt >= xout_end - MIN_LOW_LEN) {

```

c/h:c

```

    if (p->score >= edge_bound && rep_len > MIN_REP_LEN) {
        fprintf(text_file, "%s %d %d end low complexity\n", name,
                p->x0, p->xt);
        /* We can either cut it out, or mask it. in the old trans.cln
        version we masked it, but I think it would be better to cut
        it out. In the meantime I will leave it to be masked out,
        so we can compare to the previous version
        if (p->x0 <= xout_start + MIN_LOW_LEN) {
            xout_start = p->xt;
        }
        else {
            xout_end = p->x0;
        }
        */
        for (k = p->x0; k <= p->xt; k++) {
            map[k] = 1;
        }
        } else {
            /* In the middle */
            if (p->score >= inner_bound && rep_len > MIN_REP_LEN) {
                fprintf(text_file, "%s %d %d middle low complexity\n",
                        name, p->x0, p->xt);
                for (k = p->x0; k <= p->xt; k++) {
                    map[k] = 1;
                }
            }
        }
        /* if the low-complexity region is inside the good umsk part -
        check for poly-A at end, or poly-T at beginning
        WARNING - we rely upon the fact that the looping of the
        low complexity regions looks only for poly-X, and uses
        the NUCL array */
        if (p->x0 <= umsk_end && p->xt >= umsk_start) {
            /* add to full low-complexity list (for umsk version output) */
            add_to_list (&full_lc_list, *p);
        }
        if (strcmp(polyst, "A") == 0 &&
            p->xt >= umsk_end - MIN_LOW_LEN &&
            p->x0 < poly_a_start) {
            poly_a_start = p->x0;
        }
        else if (strcmp(polyst, "T") == 0 &&
            p->x0 <= umsk_start + MIN_LOW_LEN &&
            p->xt > poly_t_end) {
            poly_t_end = p->xt;
        }
    }
    destroy_hilltop_list (&low_complexity_list);
}
}
}
if (bt && !mtrans) {
    fprintf (text_file, "%s not output due to BT\n", name);
    return (0);
}
else if ((xout_end - xout_start) < MIN_EST_OUT) {

```

c/h:c

A-212

```

fprintf (text_file, "%s not output since not enough remained\n\n", name);
return (0);
} else {
    if (xout_start == 0 && xout_end == len)
        fprintf (text_file, "%s uncult\n\n", name);
    else fprintf (text_file, "%s cut: %d - %d\n\n", name, xout_start,
        xout_end);

    /* Output to Crossout file - mask repeats and low complexity */
    fprintf (xout_file, ">>> %s", name, type_st);
    if (mtrans) {
        fprintf (xout_file, "%s\n", comment);
    } else {
        fprintf (xout_file, "#DT %s\n", date_st);
    }

    for (i = xout_start; i < xout_end; i++) {
        if (map[i] == 1) fprintf (xout_file, "N");
        else fprintf (xout_file, "%c", seq[i]);
        if ((i-xout_start) % 60 == 59 || i == xout_end - 1) {
            fprintf (xout_file, "\n");
        }
    }

    /* Also, output to unmasked file - without masking low complexity */
    if (mrna) { umsk_start = 0; umsk_end = len; }
    start_st[0] = end_st[0] = '\0';
    if (umsk_start != 0) printf (start_st, "#BE %d", umsk_start);
    if (umsk_end != len) printf (end_st, "#EN %d", umsk_end);
    printf (nl_st, "NL %d", umsk_end - umsk_start);
    fprintf (unmasked_file, ">>> %s %s\n", name, type_st, comment,
        start_st, end_st, nl_st);

    /* generate repeat and vector header information - The repeat locations
    are relative to the start of the umsk version (but never out of it's
    bounds), the vector locations are absolute (relative to the original
    version) */

    for (p = list; p != NULL; p = p->next) {
        /* if (p->score >= pbound) { ADD ONLY SIGNIFICANT REPEATS? */
        if (p->type) {
            fprintf (unmasked_file, "#RE %d-%d", (p->x0 < umsk_start ? 0 : p->x0-
                umsk_start),
                (p->xt > umsk_end ? umsk_end-umsk_start : p->xt-umsk_start));
        }
        /*) FOR SIGNIFICANT REPEATS? */

        for (p = vector_list; p != NULL; p = p->next) {
            fprintf (unmasked_file, "#VE %d-%d", p->x0, p->xt);
        }

        /* generate low complexity information */
        for (p = full_lc_list; p != NULL; p = p->next) {
            fprintf (unmasked_file, "#LC %d-%d", (p->x0 < umsk_start ? 0 : p->x0-
                umsk_start),
                (p->xt > umsk_end ? umsk_end-umsk_start : p->xt-umsk_start));
        }
    }

```

clin.c

```

}

/* print start of poly-A at end (if exists), and end of poly-T at beginning
(if exists) */
if (poly_a_start < len) {
    fprintf (unmasked_file, "#PA %d", poly_a_start);
}

if (poly_t_end > -1) {
    fprintf (unmasked_file, "#PT %d", poly_t_end);
}

/* print end of quality place */
if (q > 0) {
    fprintf (unmasked_file, "#QL %d", (q > umsk_end ? umsk_end-umsk_start :
        q-umsk_start));
}

/* print cut point based upon Ns */
if (n_pos < umsk_end) {
    fprintf (unmasked_file, "#NC %d", n_pos-umsk_start);
}

fprintf (unmasked_file, "\n");

/* print unmasked version */
for (i = umsk_start; i < umsk_end; i++) {
    fprintf (unmasked_file, "%c", seq[i]);
    if ((i-umsk_start) % 60 == 59 || i == umsk_end - 1) {
        fprintf (unmasked_file, "\n");
    }
}

fprintf (text_file, "\n");
fflush (text_file);
fflush (unmasked_file);
fflush (xout_file);
return (xout_end - xout_start);
}

/*=====
int n_cut (char* seq)
{
    int i, j, bad = 0, frame_bad, frame=3*FRAME, len;

    /* count N's in last frame */
    len = strlen(seq);
    for (i = len-1; i >= len-FRAME; i--) if (seq[i]!='N') bad++;

    /* Search for last bad position in sequence */
    while ((i > 0) && (frame > 0)) {
        if (seq[i]!='N') bad++;
        if (seq[i+FRAME] == 'N') bad--;
    }
}
=====

```

clin.c

```
/* Dev Stage */
i = look_for ("DS ", comment, 0);
if (i >= 0) {
    j = look_for ("#", comment, i+1);
    dev_stage = subseq (comment, i+4, j < 0 ? len : j-2);
    else dev_stage = 0;
}

/* Insert Size */
i = look_for ("IS ", comment, 0);
if (i >= 0) {
    j = look_for ("#", comment, i+1);
    insert_size = subseq (comment, i+4, j < 0 ? len : j-2);
    else insert_size = 0;
}

/* Date */
i = look_for ("DT ", comment, 0);
j = look_for (" ", comment, i+4);
date = subseq (comment, i+4, j < 0 ? len : j-1);
strncpy (date_st, date, 50);

ch_st[0] = li_st[0] = ti_st[0] = sx_st[0] = '0';
cl_st[0] = dr_st[0] = ds_st[0] = is_st[0] = '0';
if (chromosome != 0) sprintf (ch_st, "CH %s", chromosome);
if (library != 0) sprintf (li_st, "LB %s", library);
if (tissue != 0) sprintf (ti_st, "TI %s", tissue);
if (sex != 0) sprintf (sx_st, "SX %s", sex);
if (clone != 0) sprintf (cl_st, "CL %s", clone);
if (direction != 0) sprintf (dr_st, "DR %s", direction);
if (dev_stage != 0) sprintf (ds_st, "DS %s", dev_stage);
if (insert_size != 0) sprintf (is_st, "IS %s", insert_size);

sprintf (comment, "LN %d #AC %s #NI %s #OS %s #DE %s%s%s%s%s %DT %s",
        len_seq, accession, nid, organism, definition, ch_st, li_st,
        ti_st, sx_st, cl_st, dr_st, ds_st, is_st, date);

free (definition);
free (accession);
free (nid);
free (organism);
free (length);
free (date);
if (chromosome != 0) free (chromosome);
if (library != 0) free (library);
if (tissue != 0) free (tissue);
if (sex != 0) free (sex);
if (clone != 0) free (clone);
if (direction != 0) free (direction);
if (dev_stage != 0) free (dev_stage);
if (insert_size != 0) free (insert_size);
}

/*=====
void remove_comment_name (char *comment)
{
    char* new_comment;

    int i;

    i = look_for (" ", comment, 0);
    if (i == -1) {
        =====*/
```

cln.c

```

i = 0;
}

if ( ( new_comment = subseq (comment, i, strlen (comment)) ) != NULL ) {
    sprintf (comment, "%s", new_comment);
    free (new_comment);
}

/*=====
int look_for (char *pat, char *st, int start)
{
    int i, j, flag;

    for (i = start; i < strlen (st); i++) {
        if (st[i] != pat[0]) continue;

        /* Check if pattern starts at place i */
        for (j = flag = 1; j < strlen (pat); j++) {
            if (st[i+j] != pat[j]) {
                flag = 0;
                continue;
            }
            if (flag == 1) return (i);
        }
        return (-1);
    }

    /*
    $Log: cln.c,v $
    * Revision 1.8 1998/06/01 11:21:03 prod
    * fix error in no_cut_n (removed illegal !).
    *
    * Revision 1.7 1998/05/28 15:36:23 avir
    * added flag no_cut_n - to allow running cleaning without cutting on N
    * information in the end.
    *
    * Revision 1.6 1998/04/16 10:04:51 avir
    * combined the cleaning program of the ESTs/RNAs with the
    * cleaning program of the transcripts.
    * Added a flag for running with transcripts.
    * Ignore vector files for transcripts.
    * Repeats in transcripts must be significantly long.
    * Don't look for quality info in header of transcripts.
    * Create different comment line for the transcript.
    * Don't cut transcripts on Ns information or on vector information.
    * Look for POLY-X in transcripts (apart from poly-X).
    *
    * Revision 1.5 1998/04/14 07:28:06 prod
    * Add indication if found poly-A at end, or poly_T ant start
    *
    * Revision 1.4 1998/04/13 10:03:25 prod
    * change the full path of prm.h to relative path.
    *
    * Revision 1.3 1998/04/13 09:35:13 prod
    * added ver flag
    *
    * Revision 1.2 1998/04/13 09:23:34 prod
    *=====*/
```

cln.c

```

if (bad < MANY_N) frame--;
else frame=2*FRAME;
i--;
}
if ((i == 0) && (frame == 4*FRAME - len + 1)) i = len;
/* Either the whole sequence is bad, or after i we have two good frames
and then bad data */
bad = 0;
if (frame == 0 && i != len) i += 2*FRAME;
/* count N's at the first frame of the sequence */
for (j = 0, j < FRAME; j++) if (seq[j] == 'N') bad++;
/* Keep advancing through the beginning of the sequence */
frame_bad = bad;
for (; j <= i; j++) {
    if (seq[j] == 'N') {
        bad++;
        frame_bad++;
    }
    if (seq[j-FRAME] == 'N') frame_bad--;
    if (frame_bad >= REALLY_BAD) {
        /* There is a really bad frame at the beginning of the sequence - move
        i back to there and exit this loop */
        i = j - FRAME;
        bad = bad - frame_bad;
    }
}
/* Global check of N's percentage at the beginning. If it is too high -
throw the sequence away */
if ((i == len) && (bad >= MANY_N)) i = 0;
if (bad*100 > (i + 1)*PERCENTAGE) i = 0;
/* So far i went back at least 2 frames. If it is exactly 2 frames then
we don't want to clip */
if (i != len - 2*FRAME - 1) return (i);
else return (len);
}
/*=====
void parse_comment (char *comment, int len_seq, char *date_st)
{
    char *definition, *accession, *nid, *organism, *length;
    char *chromosome, *library, *tissue, *clone, *date;
    char *dev_stage, *insert_size, *direction, *sex;
    char ch_st[LINE_SIZE], li_st[LINE_SIZE], ti_st[LINE_SIZE], cl_st[LINE_SIZE];
    char sx_st[LINE_SIZE], dr_st[LINE_SIZE], ds_st[LINE_SIZE], is_st[LINE_SIZE];
    int i, j, len;

    len = strlen (comment);
    /* Accession number */
    i = look_for ("#AC ", comment, 0);
    j = look_for (" ", comment, i+4);
    accession = subseq (comment, i+4, j < 0 ? len : j-1);
    /* NID */
    i = look_for ("#NI ", comment, 0);

```

c/n.c

```

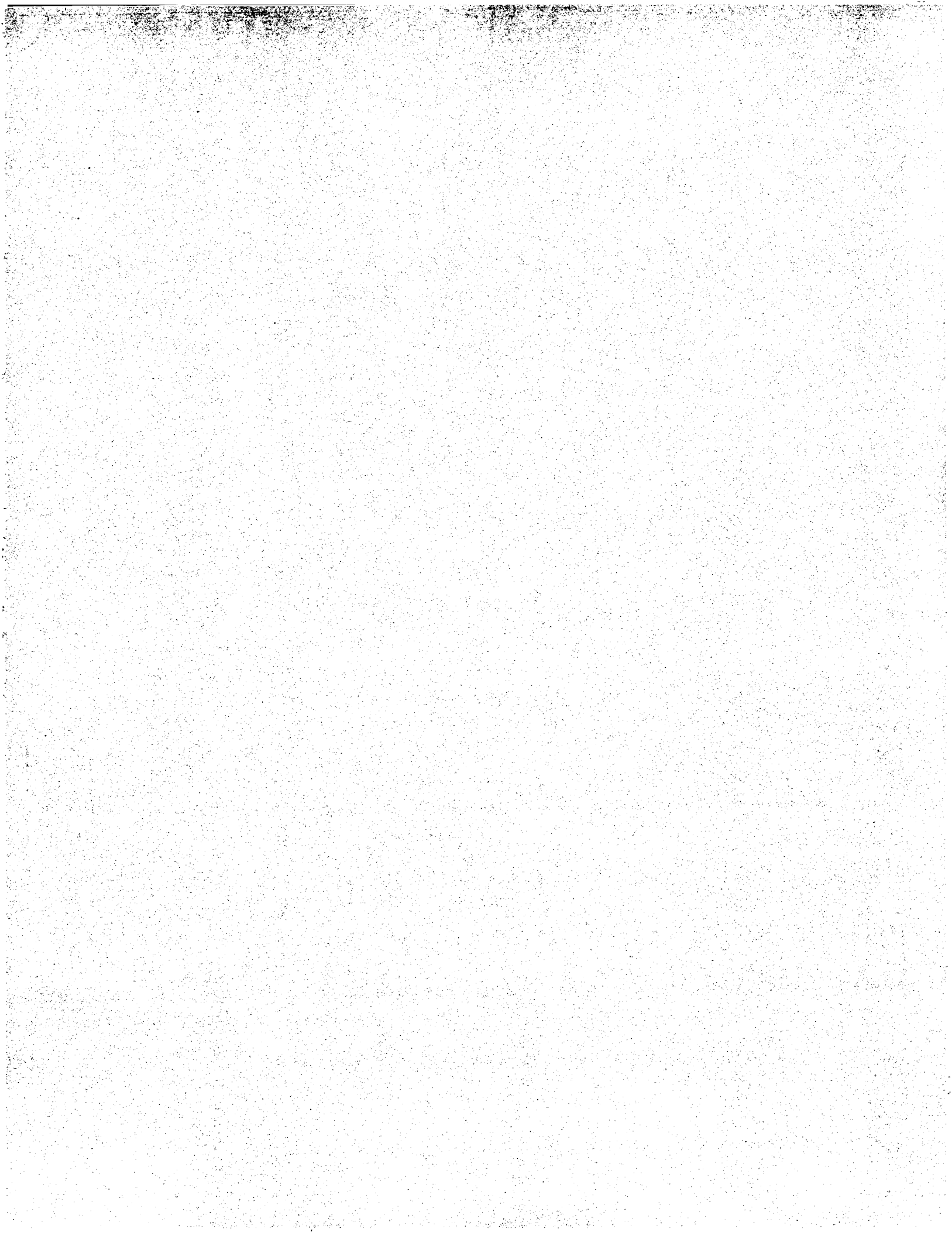
j = look_for (" ", comment, i+4);
nid = subseq (comment, i+4, j < 0 ? len : j-1);
/* Organism */
i = look_for ("#OS ", comment, 0);
j = look_for ("#", comment, i+4);
organism = subseq (comment, i+4, j < 0 ? len : j-2);
/* Definition: from here, until #LN */
i = look_for ("#DE ", comment, 0);
j = look_for ("#", comment, i+4);
definition = subseq (comment, i+4, j < 0 ? len : j-2);
/* Length */
i = look_for ("#LN ", comment, 0);
j = look_for (" ", comment, i+4);
length = subseq (comment, i+4, j < 0 ? len : j-1);
/* Chromosome */
i = look_for ("#CH ", comment, 0);
if (i >= 0) {
    j = look_for ("#", comment, i+1);
    chromosome = subseq (comment, i+4, j < 0 ? len : j-2);
    else chromosome = 0;
}
/* Library */
i = look_for ("#LI ", comment, 0);
if (i >= 0) {
    j = look_for ("#", comment, i+1);
    library = subseq (comment, i+4, j < 0 ? len : j-2);
    else library = 0;
}
/* Tissue */
i = look_for ("#TI ", comment, i);
if (i >= 0) {
    j = look_for ("#", comment, i+1);
    tissue = subseq (comment, i+4, j < 0 ? len : j-2);
    else tissue = 0;
}
/* Sex */
i = look_for ("#SX ", comment, 0);
if (i >= 0) {
    j = look_for ("#", comment, i+1);
    sex = subseq (comment, i+4, j < 0 ? len : j-2);
    else sex = 0;
}
/* Clone */
i = look_for ("#CL ", comment, 0);
if (i >= 0) {
    j = look_for ("#", comment, i+1);
    clone = subseq (comment, i+4, j < 0 ? len : j-2);
    else clone = 0;
}
/* Direction */
i = look_for ("#DR ", comment, 0);
if (i >= 0) {
    j = look_for ("#", comment, i+1);
    direction = subseq (comment, i+4, j < 0 ? len : j-2);
    else direction = 0;
}

```

c/n.c

A-216

```
* - separate xout and umsk begin/end points
* - print to umsk header:
* 1. ALL repeats found.
* 2. vectors used to cut the umsk version
* 3. ALL low complexity places found
* 4. place where quality cutting would have been done.
* 5. place where NS cutting would have been done.
*
*/
```

Sun Aug 9 10:39:55 1998

Listing for Adam Sartiel

Sun Aug 9 10:39:55 1998

Listing for Adam Sartiel

```

/*
 * Copyright 1996 Compugen, Ltd..
 * Authorization to use this code is given solely to Compugen customers.
 * This authorization is limited by the terms of the Compugen Hardware and
 * Software License Agreement.
 *
 * Last update: $date: 1998/04/12 15:00:16 $ by $Author: prod $
 * Revision: $Revision: 1.1 $
 */
#include <stdio.h>
#include <strings.h>
#include <sys/types.h>
#include <malloc.h>
#include <math.h>

#include "hilltops.h"

/*=====*/
char *subseq (char *seq, int first, int last)
/*
 * Cut a sub-sequence from a given input sequence. If first > last, flip it
 * and translate C<->G, A<->T, (and subsets as well).
 */
{
    char *res;
    int i, len, newlen, flip = 0;

    len = strlen (seq);

    if (first < 0) first = 0;
    if (first >= len) first = len-1;
    if (last < 0) last = 0;
    if (last >= len) last = len-1;

    newlen = last - first;
    if (newlen < 0) {
        flip = 1;
        newlen = -newlen;
    }
    newlen++;
    if (!res = (char *) malloc (newlen+1))
        error(1, -1, "subseq: can't allocate string\n");
    res[newlen] = '\0';
    if (flip)
        for (i = 0; i < newlen; i++)
            res[i] = inv(seq[first-i]);
    else
        for (i = 0; i < newlen; i++)
            res[i] = seq[first+i];
    return (res);
}

/*=====*/
char inv (char c)
/*
 * return base pair
 */

```

hilltops.c

```

(
    char alphabet[] = "ACMGRSVTWYHKDBN";
    /* K = G or T M = A or C R = G or A S = G or C W = A or T Y = T or C
     * B = G or T or C D = G or A or T H = A or C or T V = G or C or A
     * N = A or C or G or T */
    if (c == 'A') return 'T';
    if (c == 'C') return 'G';
    if (c == 'G') return 'C';
    if (c == 'T') return 'A';
    if (c == '-') return '-';
    if (c == 'N') return 'N';
    if (c == 'R') return 'Y';
    if (c == 'Y') return 'R';
    if (c == 'K') return 'M';
    if (c == 'M') return 'K';
    if (c == 'S') return 'S';
    if (c == 'W') return 'W';
    if (c == 'B') return 'V';
    if (c == 'D') return 'H';
    if (c == 'H') return 'D';
    if (c == 'V') return 'B';
    if (c == 'X') return 'X';
    return c;
)

/*=====*/
void create_segment (segment **obj)
{
    if (!((*obj) = (segment *) malloc (sizeof(segment))))
        error(1, -1, "couldn't allocate memory for segment.\n");

    (*obj)->next = NULL;
    (*obj)->start = 0;
    (*obj)->end = 0;
    (*obj)->score = 0.0;
    (*obj)->best_y = 0;
    (*obj)->x0 = 0;
    (*obj)->y0 = 0;
}

/*=====*/
void destroy_segment (segment **obj)
{
    free (*obj);
    *obj = NULL;
}

/*=====*/
void destroy_segment_list (segment **obj)
{
    segment *segp1, *segp2;

    for (segp1 = *obj; segp1 != NULL; ) {
        segp2 = segp1->next;
        destroy_segment (&segp1);
        segp1 = segp2;
    }
    *obj = NULL;
}

/*=====*/

```

hilltops.c

```

void copy_segment (segment from, segment *obj)
{
    obj->next = from->next;
    obj->start = from->start;
    obj->end = from->end;
    obj->score = from->score;
    obj->best_y = from->best_y;
    obj->x0 = from->x0;
    obj->y0 = from->y0;
}

/*=====*/
void create_hilltop (hilltop **obj)
/*
 * Initialize a sequence alignment structure
 */
{
    if (!(*obj) = (hilltop *) malloc (sizeof(hilltop)))
        error(1, -1, "couldn't allocate memory for hilltop.\n");

    (*obj)->next = NULL;
    (*obj)->type = UNDEFINED;
    (*obj)->final = 0;
    (*obj)->len = 0;
    (*obj)->x0 = 0;
    (*obj)->y0 = 0;
    (*obj)->xt = 0;
    (*obj)->yt = 0;
    (*obj)->score = 0.0;
    (*obj)->area = 0;
    (*obj)->id_percent = 0.0;
    (*obj)->sim_percent = 0.0;

    (*obj)->name[0] = '\0';
    (*obj)->boundary = NULL;
    (*obj)->x = NULL;
    (*obj)->y = NULL;
    (*obj)->diff = NULL;
}

/*=====*/
void destroy_hilltop (hilltop **obj)
{
    segment *seggp, *seggp2;

    if ((*obj)->x != NULL) free ((*obj)->x);
    if ((*obj)->y != NULL) free ((*obj)->y);
    if ((*obj)->diff != NULL) free ((*obj)->diff);
    destroy_segment_list (&((*obj)->boundary));

    free (*obj);
    *obj = NULL;
}

/*=====*/
void destroy_hilltop_list (hilltop **obj)
{
    hilltop *htpl, *htp2;

    for (htpl = *obj; htpl != NULL; ) {
        htp2 = htpl->next;

```

hilltops.c

```

    destroy_hilltop (&htpl);
    htpl = htp2;
}

*obj = NULL;

/*=====*/
void copy_hilltop (hilltop from, hilltop *obj)
/*
 * Copy a sequence alignment structure
 */
{
    segment *seggp, **seggpp;

    obj->next = from->next;
    obj->type = from->type;
    obj->final = from->final;
    obj->len = from->len;
    obj->x0 = from->x0;
    obj->y0 = from->y0;
    obj->xt = from->xt;
    obj->yt = from->yt;
    obj->score = from->score;
    obj->area = from->area;
    obj->id_percent = from->id_percent;
    obj->sim_percent = from->sim_percent;
    strcpy (obj->name, from->name);

    if (! (obj->x = (char *) realloc (obj->x, 1*from->len)))
        error(1, -1, "couldn't allocate memory for hilltop.\n");
    if (from->x) strcpy (obj->x, from->x);

    if (! (obj->y = (char *) realloc (obj->y, 1*from->len)))
        error(1, -1, "couldn't allocate memory for hilltop.\n");
    if (from->y) strcpy (obj->y, from->y);

    if (! (obj->diff = (char *) realloc (obj->diff, 1*from->len)))
        error(1, -1, "couldn't allocate memory for hilltop.\n");
    if (from->diff) strcpy (obj->diff, from->diff);

    seggp = &(obj->boundary);
    for (seggp = from->boundary; seggp != NULL; seggp = seggp->next) {
        create_segment (*seggp);
        copy_segment (*seggp, *seggpp);
        seggp = &((*seggp)->next);
    }

    /*=====*/
    void update_all_hilltops (segment *seg_list, hilltop **ht_list, int x, int w)
/*
 * Use the knowledge of the previous column and the segments of the
 * current column to update the area-map of all locations above cutoff.
 * "w" is used for wraparound. If not zero, hilltops wrap around the
 * horizontal edge of the matrix. In this case w should be equal to "leny".
 */
{
    int a, b, c, d, delete_flag;
    segment *segpl, *seggp2, *old_boundary, **seggpp;
    hilltop *htpl, *htp2, **htpp;

```

hilltops.c

```

/* First, bring segments to correct form (separate x0 and y0) */
for (segl = seg_list; segl != NULL; segl = segl->next) {
    segl->y0 = (segl->x0 & 0xffff);
    segl->x0 >>= 16;
    for (; x - segl->x0 > (1 << 16); segl->x0 += (1 << 16));
    for (; segl->best_y - segl->y0 > (1 << 16); segl->y0 += (1 << 16));
}

/* Go over the hilltops (which are updated to the previous column) */
for (htp = *ht_list; htp != NULL; htp = htp->next) {
    /* cut the boundary information out from the hilltop */
    old_boundary = htp->boundary;
    htp->boundary = NULL;

    /* In a hilltop, go over the segments it had in the previous column */
    for (segl = old_boundary; segl != NULL; segl = segl->next) {
        a = segl->start;
        b = segl->end;
        /* Now, try to find a segment in the new column which overlaps */
        for (seggp = &seg_list; seggp != NULL; ) {
            c = (*seggp)->start;
            d = (*seggp)->end;
            if ((c <= b+1 && d >= a) || (w && (c == 0) && (b == w-1))) {
                /* Overlap : note the exact condition! */
                update_hilltop (htp, *seggp, x);
                /* Delete the (used) segment from the list */
                segp2 = (*seggp);
                (*seggp) = ((*seggp)->next);
                destroy_segment (&seggp2);
            } else /* Only if we don't delete - advance */
                seggp = &((*seggp)->next);
        }
    }
}

```

```

/* We now have the new hilltop boundary, unless it should be merged
with a preceding hilltop. Check them all */
for (segl = old_boundary; segl != NULL; segl = segl->next) {
    a = segl->start;
    b = segl->end;
    for (htpp = ht_list; *htpp != htp; ) {
        delete_flag = 0;
        for (segg2 = (*htpp)->boundary; segg2 != NULL; segg2 = segg2->next) {
            c = segg2->start;
            d = segg2->end;
            if (c <= b+1 && d >= a) { /* Overlap : merge the hilltops */
                merge_hilltops (htp, *htpp);
                /* Delete the merged hilltop from the list */
                htp2 = (*htpp);
                (*htpp) = ((*htpp)->next);
                destroy_hilltop (&htpp2);
                delete_flag = 1;
                break;
            }
        }
        /* Only if we don't delete - advance */
        if (!delete_flag) htp = &((*htpp)->next);
    }
    destroy_segment_list (&old_boundary);
}

```

```

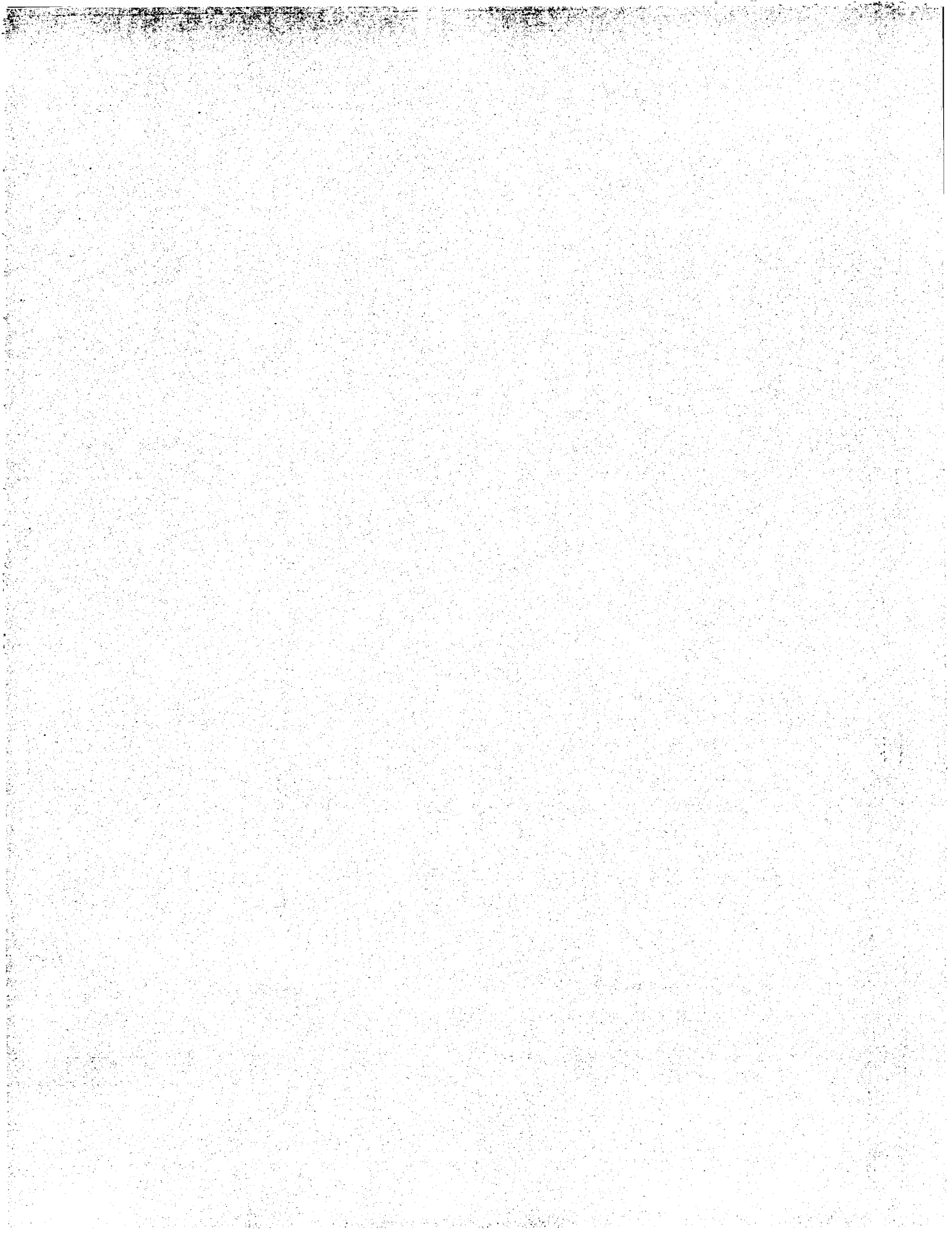
)
/* create new records for segments which are still unassociated */
for (segl = seg_list; segl != NULL; segl = segl->next) {
    create_hilltop (&htp);
    update_hilltop (htp, *segl, x);
    htp->next = *ht_list;
    *ht_list = htp;
}
destroy_segment_list (&seg_list);
)
/*=====
void update_hilltop (hilltop *htp, segment seg, int x)
/*
* We have ascertained that seg is associated with rec. We now have to
* update the data in rec according to seg.
*/
{
    segment *seggp;

    htp->area += seg.end - seg.start + 1;
    if (htp->score < seg.score) {
        htp->score = seg.score;
        htp->xt = x;
        htp->yt = seg.best_y;
        htp->x0 = seg.x0;
        htp->y0 = seg.y0;
    }
    create_segment (&seggp);
    copy_segment (seg, seggp);
    seggp->next = htp->boundary;
    htp->boundary = seggp;
}
}

/*=====
void merge_hilltops (hilltop *htpl, hilltop *htp2)
/*
* Merge the two areas into the first one.
*/
{
    segment *seggp;

    htpl->area += htp2->area;
    if (htpl->score < htp2->score) {
        htpl->score = htp2->score;
        htpl->xt = htp2->xt;
        htpl->yt = htp2->yt;
        htpl->x0 = htp2->x0;
        htpl->y0 = htp2->y0;
    }
    for (seggp = htp2->boundary; seggp->next != NULL; seggp = seggp->next);
    seggp->next = htpl->boundary;
    htp2->boundary = NULL;
}
/*=====

```



```

/* Copyright 1996 Compugen, Ltd.
 * Authorization to use this code is given solely to Compugen Customers.
 * This authorization is limited by the terms of the Compugen Hardware and
 * Software License Agreement.
 *
 * Last update: $Date: 1998/04/12 15:00:04 $ by $Author: prod $
 * Revision: $Revision: 1.1 $
 */

```

```

#include <stdio.h>
#include <strings.h>
#include <sys/types.h>
#include <malloc.h>
#include <math.h>

```

```

#include "dp.h"

```

```

/* Macros */
#define MAT(i,j) (mat + ((i)*leny + (j)))

```

```

/*Definition of constants */

```

```

#define INF 100000

```

```

#define NONE -1
#define DIAG 0
#define LEFT 1
#define DOWN 2
#define XGAP 3
#define YGAP 4
#define ALT 5

```

```

/*TypeDefs */

```

```

typedef struct node {
    double match, ins, del;
} node;

```

```

typedef struct lnode {
    double sc[3];
    int x0[3], y0[3], s0[3];
} lnode;

```

```

typedef struct sixnode {
    double match, ins, del, xgap, ygap, alt;
} sixnode;

```

```

typedef struct lsixnode {
    double sc[6];
    int x0[6], y0[6], s0[6];
} lsixnode;

```

```

/* static variables */

```

```

int X_GO, X_GE, Y_GO, Y_GE;
int sc[26][26]; /* ALPHABET is a subset of the English Alphabet */
double exact_sc[26][26];

```

```

void dbg_print (char x[], char y[], node *mat);
void get_match (char x[], char y[], node *mat, int it, int jt, hilltop *ht);
void get_six18_match (char x[], char y[], sixnode *mat, int it, int jt, int st,

```

```

void call_linear_six18_alignment (char x[], char y[], hilltop *ht);
void linear_six18_alignment (char x[], char y[], hilltop *ht, int bs, int es);
void improve_six18_result (hilltop *ht, int mode);
void get_band_match (char x[], char y[], int x0, int y0, int width, node *mat,
    int it, int jt, hilltop *ht);

```

```

/*=====
void init_matrix (int match, int mismatch, int tGB, int tGO)
*/

```

```

/* Initialize static variables for nucleotides. we use the following
 * nomenclature:

```

```

 * K = G or T M = A or C R = G or A S = G or C W = A or T Y = T or C
 * B = G or T or C D = G or A or T H = A or C or T V = G or C or A
 * N = A or C or G or T X = None of the above.
 */

```

```

int i, j, len, good, bad;
char alphabet[] = "XACMGRSVTWYHKDBN";
int pop[16] = {0, 1, 1, 2, 1, 2, 1, 2, 3, 1, 2, 2, 3, 2, 3, 4};
double maxc, minc, val, c, p, q, exp(), log();
/*

```

```

 * Approximate the multiplicative constant c used to compute the
 * match and mismatch user-controlled constants
 */

```

```

if (match <= 0 || mismatch >= 0 || match > (-3*mismatch))
    error(1, -1, "Bad values for match (%d) and mismatch (%d)",
        match, mismatch);
maxc = 100000.0;
minc = 0.1;
for (i = 0; i < 100; i++) {
    c = (maxc + minc) / 2;
    val = (exp((double)match/c)+3*exp((double)mismatch/c))/4;
    if (val < 1.0) maxc = c;
    else minc = c;
}

```

```

p = exp(match/c)/16;
q = exp(mismatch/c)/16;
/* printf ("p = %g, q = %g, c = %g\n", p, q, c); */

```

```

for (i = 0; i < 26; i++)
    for (j = 0; j < 26; j++)
        sc[i][j] = exact_sc[i][j] = mismatch;

```

```

/*
 * For each letter of the alphabet, find the correct score by
 * aposteriori calculation, using p, q, and c.
 */

```

```

for (i = 1; i < 16; i++)
    for (j = 1; j < 16; j++) {
        good = pop[i & j];
        bad = pop[i] * pop[j] - good;
        val = c * log(16 * (good * p + bad * q) / (good + bad));
        exact_sc[alphabet[i] - 'A'][alphabet[j] - 'A'] = val;
        sc[alphabet[i] - 'A'][alphabet[j] - 'A'] = (int) (val + 0.5 * (val > 0));
    }

```

```

/*
for (i = 0; i < 26; i++) {

```

```

for (j = 0; j < 26; j++) {
    if (exact_sc[i][j] >= 0) printf (" ");
    printf ("%2.2f ", exact_sc[i][j] / 100);
}
printf ("\n");
}
fflush (stdout); /* */
X_GO = tGO;
X_GE = tGE;
Y_GO = tGO;
Y_GE = tGE;
}

/* ===== */
void get_alignment (char in_x[], char in_y[], hilltop *ht, int where)
/*
 * Dynamic programming routine to match two nucleotide sequences, using
 * square memory.
 * From the contents of "ht" we figure out whether to compute the whole matrix
 * or to cut above a certain diagonal:
 *
 * |*****| the given value of loop_end is the
 * |*****| height of the vertical bar in the drawing.
 *
 * The "where" parameter can be either BEST (find best score) or CORNER (force
 * the routine to take the upper right corner).
 */
{
    int i, j, best_i = 0, best_j = 0, tmp, id_count, sim_count, no_n_count;
    int lenx, leny, mode, loop_end;
    double max_score = 0.0;
    node *diag, *left, *down, *current, *mat, zero;
    hilltop *res;
    char *x, *y;

    /* copy input strings to working strings, shorten if necessary */
    x = subseq (in_x, ht->x0, ht->xt); lenx = strlen (x);
    y = subseq (in_y, ht->y0, ht->yt); leny = strlen (y);

    if (lenx*leny > MAX_FULL*MAX_FULL) {
        /* We need a linear memory alignment here (which is slower) */
        linear_alignment (in_x, in_y, ht, 0, 0);
        free (x);
        free (y);
        ht->final = 1;
        ht->len = strlen (ht->x);
        id_count = sim_count = no_n_count = 0;
        for (i = 0; i < strlen (ht->diff); i++) {
            id_count += (ht->diff[i] == '|');
            if (ht->x[i] != '-' && ht->y[i] != '-')
                sim_count += (exact_sc[ht->x[i]-'A'][ht->y[i]-'A'] > 0);
            if (ht->x[i] != 'N' && ht->y[i] != 'N') no_n_count++;
        }
        ht->id_percent = 100*(double)(id_count) / (double)(no_n_count);
        ht->sim_percent = 100*(double)(sim_count) / (double)(no_n_count);
        return;
    }

    if (ht->type == TANDEM) {

```

```

loop_end = ht->x0 - ht->y0;
if (loop_end > leny) loop_end = leny;
} else {
    loop_end = leny;
}

/* allocate memory and initialize matrix boundaries */
if (!mat = (node *) malloc ((1+lenx*leny * sizeof (node))))
    error(1, -1, "couldn't allocate memory for nodes 1 (%d %d) \n",
        lenx, leny);

zero.match = zero.ins = zero.del = 0;
for (j = 0; j < lenx; j++) {
    if (j + loop_end >= leny) break;
    if (j + loop_end < 0) continue;
    current = &mat[j*leny + j+loop_end];
    *current = zero;
}

/* main loop */
for (i = 0; i < lenx; i++) {
    /* Advance one column and initialize pointers */
    current = &mat[i*leny];
    if (i == 0) left = &zero;
    else left = &mat[(i-1)*leny];
    diag = &zero; down = &zero;

    for (j = 0; j < loop_end; j++) {
        current->match = diag->match;
        if (diag->ins > current->match) current->match = diag->ins;
        if (diag->del > current->match) current->match = diag->del;
        current->match += exact_sc[x[i]-'A'][y[j]-'A'];
        if (current->match < 0) current->match = 0;

        current->ins = left->match - X_GO;
        if (left->ins - X_GE > current->ins) current->ins = left->ins - X_GE;
        if (current->ins < 0) current->ins = 0;

        current->del = down->match - Y_GO;
        if (down->del - Y_GE > current->del) current->del = down->del - Y_GE;
        if (current->del < 0) current->del = 0;

        /* Save best node */
        if ((current->match) >= max_score) {
            max_score = current->match; best_i = i; best_j = j;
        }
        /* Advance pointers for next node */
        if (i > 0) {diag = left; left++;}
        down = current; current++;
    }
    if (loop_end < leny) loop_end++;
}
if (where == CORNER) {
    best_i = lenx-1;
    best_j = leny-1;
}
/*for (j = leny - 1; j >= 0; j--) {
    for (i = 0; i < lenx; i++)
        printf ("%2d ", mat[i*leny+j].match);
    printf ("\n");
}*/

```

```

for (i = 0; i < lenx; i++)
    printf ("%2d %2d ", mat[i*leny+j].ins, mat[i*leny+j].del);
    printf ("\n\n");
} /**/
get_match (x, y, mat, best_i, best_j, ht);
free (mat);
free (x);
free (y);
}

/*=====
void get_match (char x[], char y[], node *mat, int it, int jt, hilltop *ht)
/* Extract the best solution */
{
    char *nice_x, *nice_y, *nice_diff;
    int last, i, j, loc, end, id_count, sim_count, no_n_count, lenx, leny;
    double tmp;
    node *current, *diag, *left, *down;

    lenx = strlen (x);
    leny = strlen (y);

    end = it + jt;
    if (((nice_x = (char *)malloc (end+2)) == NULL) ||
        ((nice_y = (char *)malloc (end+2)) == NULL) ||
        ((nice_diff = (char *)malloc (end+2)) == NULL))
        error (1, -1, "Can't allocate memory for nice_strings");
    nice_x[end+1] = nice_y[end+1] = nice_diff[end+1] = '\0';
    i = it; j = jt;
    last = DIAG;

    for (loc = end; last != NONE; loc--) {
        current = &mat[i*leny + j];
        left = &mat[(i-1)*leny + j];
        down = &mat[i*leny + j-1];
        diag = &mat[(i-1)*leny + j-1];

        if (last == DIAG) {
            nice_x[loc] = x[i];
            nice_y[loc] = y[j];
            if (x[i] == y[j] && exact_sc[x[i]-'A'][y[j]-'A'] > 0)
                nice_diff[loc] = '|';
            else nice_diff[loc] = '-';
            tmp = current->match - exact_sc[x[i]-'A'][y[j]-'A'];
            if (tmp == 0) last = NONE;
            else {
                if (fabs(diag->match - tmp) < EPSILON) last = DIAG;
                else if (fabs(diag->del - tmp) < EPSILON) last = DIAG;
                else if (fabs(diag->ins - tmp) < EPSILON) last = LEFT;
                else printf ("error in back-tracking 1, %d %d\n", i, j);
            } if (last == NONE) break;
            i--; j--;
        } else if (last == LEFT) {
            nice_x[loc] = x[i];
            nice_y[loc] = '-';
            nice_diff[loc] = '-';
            if (current->ins == 0) last = NONE;
            else {
                tmp = current->ins;
            }
        }
    }
}

```

```

if (fabs(left->match - tmp-X_GO) < EPSILON) last = DIAG;
else if (fabs(left->ins - tmp-X_GE) < EPSILON) last = LEFT;
else printf ("error in back-tracking 2, %d %d\n", i, j);
} i--;
} else if (last == DOWN) {
    nice_x[loc] = '-';
    nice_y[loc] = y[j];
    nice_diff[loc] = '-';
    if (current->del == 0) last = NONE;
    else {
        tmp = current->del;
        if (fabs(down->del - tmp-Y_GE) < EPSILON) last = DOWN;
        else if (fabs(down->match - tmp-Y_GO) < EPSILON) last = DIAG;
        else printf ("error in back-tracking 3, %d %d\n", i, j);
    } j--;
}
}

ht->final = 1;
ht->len = strlen (&nice_y[loc]);
ht->x = subseq (&nice_x[loc], 0, ht->len);
ht->y = subseq (&nice_y[loc], 0, ht->len);
ht->diff = subseq (&nice_diff[loc], 0, ht->len);
free (nice_x);
free (nice_y);
free (nice_diff);

id_count = sim_count = no_n_count = 0;
for (i = 0; i < strlen (ht->diff); i++) {
    id_count += (ht->diff[i] == '|');
    if (ht->x[i] != '-' && ht->y[i] != '-')
        sim_count += (exact_sc[ht->x[i]-'A'][ht->y[i]-'A'] > 0);
    if (ht->x[i] != 'N' && ht->y[i] != 'N') no_n_count++;
}
ht->id_percent = 100*(double)(id_count) / (double)(no_n_count);
ht->sim_percent = 100*(double)(sim_count) / (double)(no_n_count);
}

/*=====
void dbg_print (char x[], char y[], node *mat)
/* Debug printout of the matrix */
{
    int i, j, lenx, leny;

    lenx = strlen (x);
    leny = strlen (y);
    for (j = leny-1; j >= 0; j--) {
        printf ("%c ", y[j]);
        for (i = 0; i < lenx; i++) {
            printf ("%2d %2d ", MAT(i,j)->match,
                MAT(i,j)->ins, MAT(i,j)->del);
        }
        printf ("\n");
    }
    for (i = 0; i < lenx; i++)
        printf (" %c ", x[i]);
    printf ("\n");
}

/*=====

```



```

hilltop *HillTops (char in_x[], char in_y[], int mode, int cutoff, int bound)
/*
 * Dynamic programming routine to match two nucleotide sequences, and
 * get ALL distinct peaks, using the HillTops generalization of SW.
 * Mode = > 0 : full Smith-Waterman matrix calculation.
 * INVERTED : lower left half of matrix only (useful for inverted
 *             repeats searches)
 * STRAIGHT : lower right half of matrix only (useful for straight
 *             and tandem repeat searches).
 * Scores under bound will not be printed. between two scores above bound
 * only the better one will be printed, unless they are separated by a valley
 * with scores under cutoff.
 * The routine returns the number of alignments found. res will point to a
 * linked list of alignments of that length.
 */
{
    segment *seggp, *segtmp;
    int i, j, best_i, best_j, best, tmp, segment_flag;
    int lenx, leny, loop_end, loop_add;
    mnode *diag, *left, *down, *current, *col, *prevcol, *tmp_node;
    zero, zero2;
    char *x, *y;
    hilltop *work, *list, **htpp, *htp;

    /* copy input strings to working strings, shorten if necessary */
    x = subseq (in_x, 0, MAX_LEN); lenx = strlen (x);
    y = subseq (in_y, 0, MAX_LEN); leny = strlen (y);
    list = work = NULL;

    if ((mode == STRAIGHT || mode == INVERTED) && lenx != leny)
        printf ("Warning: Non-square matrix in triangular mode\n");

    /* allocate memory and initialize matrix vertical edge */
    if (! (col = (mnode *) malloc (1+leny * sizeof (struct mnode))) ||
        ! (prevcol = (mnode *) malloc (1+leny * sizeof (struct mnode))) ||
        error(1, -1, "couldn't allocate memory for nodes 2 (%d).\n", leny))
        return 0;

    /* Take care of triangular modes */
    loop_end = leny; loop_add = 0;
    if (mode == INVERTED) loop_add = -1;
    else if (mode == STRAIGHT) { loop_end = 0; loop_add = 1; }

    zero.match.s = zero.ins.s = zero.del.s = 0;
    zero.match.xy = zero.ins.xy = zero.del.xy = 0;
    for (j = 0; j < leny; j++) {
        col[j] = prevcol[j] = zero;
        col[j].match.xy = col[j].ins.xy = col[j].del.xy = (j+1) & 0xffff;
        /* 16 bits are for y0, 16 msbits for x0. This is enough assuming
         * no ALIGNMENT is longer than 65535 nucleotides */
    }

    /* main loop */
    for (i = 0; i < lenx; i++) {
        /* Advance one column and initialize pointers */
        tmp_node = col; col = prevcol; prevcol = tmp_node;
        left = prevcol; current = col;
        diag = &zero2; down = &zero;

        segment_flag = 0; seggp = NULL;

```

dp.c

```

/* initialize horizontal edge */
*diag = *down;
down->match.xy = down->ins.xy = down->del.xy = (((i+1) & 0xffff) << 16);

/* Loop over nodes in the column */
for (j = 0; j < loop_end; j++) {
    current->match = diag->match;
    if (diag->ins.s > current->match.s) current->match = diag->ins;
    if (diag->del.s > current->match.s) current->match = diag->del;
    current->match.s += exact_sc[i]-'A'][y[j]-'A'];
    if (current->match.s <= 0) {
        current->match.s = 0;
        current->match.xy = (((i+1) & 0xffff) << 16) ^ ((j+1) & 0xffff);
    }

    current->ins = left->match;
    current->ins.s -= X_GO;
    if (left->ins.s - X_GE > current->ins.s) {
        current->ins = left->ins; current->ins.s -= X_GE;
    }
    if (current->ins.s <= 0) {
        current->ins.s = 0;
        current->ins.xy = (((i+1) & 0xffff) << 16) ^ ((j+1) & 0xffff);
    }

    current->del = down->match;
    current->del.s -= Y_GO;
    if (down->del.s - Y_GE > current->del.s) {
        current->del = down->del; current->del.s -= Y_GE;
    }
    if (current->del.s <= 0) {
        current->del.s = 0;
        current->del.xy = (((i+1) & 0xffff) << 16) ^ ((j+1) & 0xffff);
    }

    /* Save transitions of cutoff */
    if (! (segment_flag) {
        create_segment (&seggp);
        segtmp->next = seggp;
        segtmp->start = segtmp->best_y = j; segtmp->end = loop_end-1;
        segtmp->score = current->match.s; segtmp->best_y = j;
        segtmp->x0 = current->match.xy;
        segment_flag = 1;
    }
    else {
        if (current->match.s > seggp->score) {
            seggp->score = current->match.s; seggp->best_y = j;
            seggp->x0 = current->match.xy;
        }
        if ((current->match.s) < cutoff) {
            segtmp->end = j-1; segment_flag = 0;
        }
    }

    /* Advance pointers for next node */
    diag = left; left++; down = current; current++;
}
/* Deal with segments */
update_all_hilltops (seggp, &work, i, 0);

```

dp.c

```

/* Copy boundaryless hilltops aside */
for (htpp = &work; *htpp != NULL; ) {
    if ((*htpp)->boundary == NULL) {
        htp = (*htpp);
        htp->type = mode;
        if (htp->type == STRAIGHT) && (htp->x0 - htp->yt < TANDEM_GAP))
            htp->type = TANDEM;
        (*htpp) = ((*htpp)->next);
        if (htp->score >= bound)
            trim_hilltop (in_x, in_y, htp, cutoff);
        if (htp->score >= bound) { /* Score may have dropped - check again. */
            htp->next = list;
            list = htp;
        } else destroy_hilltop (&htp);
        } else htp = &((*htpp)->next);
    }
    loop_end += loop_add;
}

/* At the end there might be some hilltops which still have boundaries */
for (htpp = &work; *htpp != NULL; ) {
    htp = (*htpp);
    htp->type = mode;
    if (htp->type == STRAIGHT) && (htp->x0 - htp->yt < TANDEM_GAP))
        htp->type = TANDEM;
    if (htp->score >= bound)
        trim_hilltop (in_x, in_y, htp, cutoff);
    if (htp->score >= bound) { /* Score may have dropped - check again. */
        destroy_segment_list (&(htp->boundary));
        htp->next = list;
        list = htp;
    } else destroy_hilltop (&htp);
}

free (col);
free (prevcol);
free (x);
free (y);
return (list);
}

/*=====
void global_alignment (char in_x[], char in_y[], int loop_end, hilltop **ht)
/*=====
* Dynamic programming routine to match two nucleotide sequences, and
* get the end of the best alignment starting at the beginning of them both.
* Returns a Hilltop structure.
* Runs in linear memory.
* Loop_end is used to cut a piece of the matrix, just like in 'get_align'.
*/
{
    segment *seggp, *seg_tmp;
    int i, j, best_i, best_j, tmp, segment_flag;
    double max_score = 0.0;
    int lenx, leny;
    node *diag, *left, *down, *current, *col, *prevcol, *tmp_node, zero, bad;
    char *x, *y;

```

dp.c

```

/* copy input strings to working strings, shorten if necessary */
x = subseq (in_x, 0, MAX_LEN); lenx = strlen (x);
y = subseq (in_y, 0, MAX_LEN); leny = strlen (y);
if (loop_end > leny) loop_end = leny;
create_hilltop (ht);

/* allocate memory and initialize matrix vertical edge */
if (!col = (node *) malloc (1+leny * sizeof (struct node))) ||
    !prevcol = (node *) malloc (1+leny * sizeof (struct node)))
    error(1, -1, "couldn't allocate memory for nodes 3 (%d).\n", leny);

zero_match = zero.ins = zero.del = 0;
bad_match = bad.ins = bad.del = -1000000;

for (j = 0; j < leny; j++) col[j] = prevcol[j] = bad;

/* main loop */
for (i = 0; i < lenx; i++) {
    /* Advance one column and initialize pointers */
    tmp_node = col; col = prevcol; prevcol = tmp_node;
    diag = &bad; left = prevcol;
    down = &bad; current = col;
    if (i == 0) diag = &zero;

    /* Loop over nodes in the column */
    for (j = 0; j < loop_end; j++) {
        current->match = diag->match;
        if (diag->ins > current->match) current->match = diag->ins;
        if (diag->del > current->match) current->match = diag->del;
        current->match += exact_sc(x[i]-y[j]-'A');
        if (current->match <= 0) current->match = bad_match;

        current->ins = left->match - X_GO;
        if (left->ins - X_GE > current->ins) current->ins = left->ins - X_GE;
        if (current->ins <= 0) current->ins = bad.ins;

        current->del = down->match - Y_GO;
        if (down->del - Y_GE > current->del) current->del = down->del - Y_GE;
        if (current->del <= 0) current->del = bad.del;

        /* Save maximal value */
        if (current->match > (*ht)->score) {
            (*ht)->score = current->match;
            (*ht)->xt = i;
            (*ht)->yt = j;
        }

        /* Advance pointers for next node */
        diag = left; left++;
        down = current; current++;
        if (loop_end < leny) loop_end++;
    }
    free (col);
    free (prevcol);
    free (x);
    free (y);
}

/*=====
double smith_waterman (char in_x[], char in_y[])
/*=====

```

dp.c

A-224

```

/*
 * Dynamic programming routine to match two nucleotide sequences, and
 * Get the end of the best local alignment.
 * Returns the maximal score.
 * Runs in linear memory.
 */
int
i, j, best_i, best_j, best;
int
lenx, leny;
double
max_score = 0.0, *help, *p;
register double tmp;
node
diag, *left, *down, *current, *col, *prevcol, *tmp_node, zero;
char
*x, *y;

/* copy input strings to working strings, shorten if necessary */
x = subseq (in_x, 0, MAX_LEN);
y = subseq (in_y, 0, MAX_LEN);
lenx = strlen (x);
leny = strlen (y);

/* allocate memory */
if (! (col = (node *) malloc ((1+leny * sizeof (struct node)))) ||
    ! (prevcol = (node *) malloc ((1+leny * sizeof (struct node)))) ||
    error (1, -1, "couldn't allocate memory for nodes 4 (%d).\n", leny);
    for (i = 0; i < lenx; i++)
        for (j = 0; j < leny; j++)
            help[i*leny+j] = exact_sc[i][y[j]]-'A';

/* initialize vertical edge */
zero.match = zero.ins = zero.del = 0;
for (j = 0; j < leny; j++) col[j] = zero;

/* main loop */
for (i = 0; i < lenx; i++) {
    /* Advance one column and initialize pointers */
    tmp_node = col; col = prevcol; prevcol = tmp_node;
    diag = &zero; left = prevcol;
    down = &zero; current = col;
    p = &help[(x[i] - 'A')*leny];

    /* Loop over nodes in the column */
    for (j = 0; j < leny; j++) {
        current->match = diag->match;
        if (diag->ins > current->match) current->match = diag->ins;
        if (diag->del > current->match) current->match = diag->del;
        /* current->match += exact_sc[i][y[j]]-'A'; */
        current->match += *p;
        if (current->match <= 0) current->match = 0;

        current->ins = left->match - X_GO;
        if (left->ins - X_GE > current->ins) current->ins = left->ins - X_GE;
        if (current->ins <= 0) current->ins = 0;

        current->del = down->match - Y_GO;
        if (down->del - Y_GE > current->del) current->del = down->del - Y_GE;
        if (current->del <= 0) current->del = 0;
    }
}

```

dp.c

```

/* Save maximal value */
if (current->match > max_score)
    max_score = current->match;
/* Advance pointers for next node */
diag = left; left++;
down = current; current++;
}
free (col);
free (prevcol);
free (x);
free (y);
free (help);
return (max_score);
}

/*-----*/
void trim_hilltop (char in_x[], char in_y[], hilltop *ht, int cutoff)
/*
 * Trim an alignment and leave just the end if the score (going back) drops
 * below 0 at any point.
 */
{
    hilltop *p;
    char *tmpx, *tmpy;
    int loop_end;

    tmpx = subseq (in_x, ht->xt, ht->xt, ht->x0);
    tmpy = subseq (in_y, ht->yt, ht->yt, ht->y0);
    loop_end = strlen (tmpx);
    if (ht->type == TANDEM || ht->type == STRAIGHT)
        loop_end = (ht->xt) - (ht->yt);
    global_alignment (tmpx, tmpy, loop_end, &p);
    if (p->score != ht->score) {
        /* printf ("trimming (%d,%d) score %d to (%d,%d) score %d.\n", ht->x0, ht->y0,
        ht->score, ht->xt - p->yt, ht->yt - p->xt, p->score); */
        ht->x0 = ht->xt - p->yt;
        ht->y0 = ht->yt - p->xt;
        ht->score = p->score;
    }
    free (tmpx);
    free (tmpy);
}

/*-----*/
void linear_alignment (char x[], char y[], hilltop *ht, int bs, int es)
/*
 * Dynamic programming routine to match two nucleotide sequences, using
 * linear memory.
 * From the contents of "ht" we figure out whether to compute the whole matrix
 * or to cut above a certain diagonal:
 *
 * -----
 * |*****| the given value of loop_end is the
 * |*****| height of the vertical bar in the drawing.
 * +-----+
 *
 * bs and es are the beginning and end states. When the routine is run from
 * the outside they should both be 0 (i.e. match state). Upon recursive calls
 * either (or both) can also be 1 (insert) or 2 (delete).
 */

```

dp.c

```

(
    int    i, j, k, x0, y0, s0, update;
    int    lenx, leny, loop_end, h_line, v_line;
    lnode  *diag, *left, *down, *current, *tmp_node;
    lnode  *col, *prevcol, zero, bad;
    hlltop *bottom, *top;

    /*printf ("In linear alignment x0=%d, xt=%d, y0=%d, yt=%d, bs=%d, es=%d\n",
        ht->x0, ht->xt, ht->y0, ht->yt, bs, es); */

    lenx = ht->xt - ht->x0 + 1;
    leny = ht->yt - ht->y0 + 1;
    /* First check if this is the end of the recursion */
    if (lenx <= 2 && leny <= 2) {
        if (!ht->x = (char *) malloc (3)))
            error(1, -1, "couldn't allocate memory for alignment string.\n");
        if (!ht->y = (char *) malloc (3)))
            error(1, -1, "couldn't allocate memory for alignment string.\n");
        if (!ht->diff = (char *) malloc (3)))
            error(1, -1, "couldn't allocate memory for alignment string.\n");
        ht->x[2] = ht->y[2] = ht->diff[2] = '\0';
        ht->diff[0] = ht->diff[1] = '-';
        ht->x[0] = x[ht->x0]; ht->y[0] = y[ht->y0];
        ht->x[1] = x[ht->xt]; ht->y[1] = y[ht->yt];

        if (bs == 0) {
            if (exact_sc[ht->x[0] - 'A'][ht->y[0] - 'A'] > 0) ht->diff[0] = '-';
            if (ht->x[0] == ht->y[0]) ht->diff[0] = '|';
        } else if (bs == 1)
            ht->y[0] = '-';
        else
            ht->x[0] = '-';

        if (es == 0) {
            if (exact_sc[ht->x[1] - 'A'][ht->y[1] - 'A'] > 0) ht->diff[1] = '-';
            if (ht->x[1] == ht->y[1]) ht->diff[1] = '|';
        } else if (es == 1)
            ht->y[1] = '-';
        else
            ht->x[1] = '-';
        ht->len = 2;
        return;
    }

    /* Define h_line and v_line */
    h_line = (lenx-1) / 2; if (lenx == 1) h_line = -1;
    v_line = (leny-1) / 2; if (leny == 1) v_line = -1;
    /* allocate memory */
    if (!col = (lnode *) malloc ((1+leny * sizeof (lnode))))
        error(1, -1, "couldn't allocate memory for nodes in linear alignment.\n");
    if (!prevcol = (lnode *) malloc ((1+leny * sizeof (lnode))))
        error(1, -1, "couldn't allocate memory for nodes in linear alignment.\n");

    zero.sc[0] = zero.sc[1] = zero.sc[2] = 0.0;
    bad.sc[0] = bad.sc[1] = bad.sc[2] = -1000000;
    for (j = 0; j < leny; j++) col[j] = prevcol[j] = bad;

    /* main loop */

```

```

for (i = 0; i < lenx; i++) {
    /* Advance one column and initialize pointers */
    if (ht->type == TANDEM) {
        loop_end = ht->x0 - ht->y0 + i; if (loop_end > leny) loop_end = leny;
    } else loop_end = leny;
    tmp_node = col; col = prevcol; prevcol = tmp_node;
    current = col; left = prevcol;
    diag = &bad; down = &bad;
    if (i == 0) diag = &zero;

    for (j = 0; j < loop_end; j++) {
        /* initialize x0, y0 and s0 */
        if ((i == h_line && j >= v_line) || (i >= h_line && j == v_line)) {
            for (k = 0; k < 3; k++) {
                current->x0[k] = i;
                current->y0[k] = j;
                current->s0[k] = k;
            }
            /* update the scores */
            update = 0;
            if (i > h_line && j > v_line) update = 1;

            /* state 0 : match */
            current->sc[0] = diag->sc[0];
            if (update) {
                current->x0[0] = diag->x0[0];
                current->y0[0] = diag->y0[0];
                current->s0[0] = diag->s0[0];
            }
            if (diag->sc[1] > current->sc[0]) {
                current->sc[0] = diag->sc[1];
                if (update) {
                    current->x0[0] = diag->x0[1];
                    current->y0[0] = diag->y0[1];
                    current->s0[0] = diag->s0[1];
                }
            }
            if (diag->sc[2] > current->sc[0]) {
                current->sc[0] = diag->sc[2];
                if (update) {
                    current->x0[0] = diag->x0[2];
                    current->y0[0] = diag->y0[2];
                    current->s0[0] = diag->s0[2];
                }
            }

            /* state 1: insert */
            current->sc[1] = left->sc[0] - X_GO;
            if (update) {
                current->x0[1] = left->x0[0];
                current->y0[1] = left->y0[0];
                current->s0[1] = left->s0[0];
            }
            if (left->sc[1] - X_GE > current->sc[1]) {
                current->sc[1] = left->sc[1] - X_GE;
                if (update) {
                    current->x0[1] = left->x0[1];
                    current->y0[1] = left->y0[1];
                    current->s0[1] = left->s0[1];
                }
            }

```

```

    }
}

/* state 2: delete */
current->sc[2] = down->sc[0] - Y_GO;
if (update) {
    current->x0[2] = down->x0[0];
    current->y0[2] = down->y0[0];
    current->s0[2] = down->s0[0];
}
if (down->sc[2] - Y_GE > current->sc[2]) {
    current->sc[2] = down->sc[2] - Y_GE;
    if (update) {
        current->x0[2] = down->x0[2];
        current->y0[2] = down->y0[2];
        current->s0[2] = down->s0[2];
    }
}
current->sc[0] += exact_sc[x[!ht->x0]-'A'][y[!ht->y0]-'A'];

if (i == 0 && j == 0)
    for (k = 0; k < 3; k++)
        if (bs == k) current->sc[k] = zero.sc[k];
    else current->sc[k] = bad.sc[k];
/* Advance pointers for next node */
diag = left; left++; down = current; current++;
}

x0 = down->x0[es] + ht->x0;
y0 = down->y0[es] + ht->y0;
s0 = down->s0[es];

create_hilltop (&bottom);
bottom->type = top->type = ht->type;
bottom->x0 = ht->x0; bottom->y0 = ht->y0;
bottom->xt = x0; bottom->yt = y0;
top->x0 = x0; top->y0 = y0;
top->xt = ht->xt; top->yt = ht->yt;
linear_alignment (x, y, bottom, bs, s0);
linear_alignment (x, y, top, s0, es);
ht->len = bottom->len + top->len - 1;

if (! (ht->x = (char *) malloc (ht->len+1)))
    error(1, -1, "couldn't allocate memory for alignment string.\n");
if (! (ht->y = (char *) malloc (ht->len+1)))
    error(1, -1, "couldn't allocate memory for alignment string.\n");
if (! (ht->diff = (char *) malloc (ht->len+1)))
    error(1, -1, "couldn't allocate memory for alignment string.\n");

for (i = 0; i < bottom->len; i++) {
    ht->x[i] = bottom->x[i];
    ht->y[i] = bottom->y[i];
    ht->diff[i] = bottom->diff[i];
}
for (i = 0; i < top->len; i++) {
    ht->x[i+bottom->len-1] = top->x[i];
    ht->y[i+bottom->len-1] = top->y[i];
    ht->diff[i+bottom->len-1] = top->diff[i];
}

```

dp.c

```

ht->x[ht->len] = ht->y[ht->len] = ht->diff[ht->len] = '\0';
ht->final = 1;

destroy_hilltop (&bottom);
free (col);
free (prevcol);
}

/*=====
int XL_GO, XL_GE, YL_GO, YL_GE;

void init_long_gaps (lGE, lGO)
{
    XL_GO = lGO;
    XL_GE = lGE;
    YL_GO = lGO;
    YL_GE = lGE;
}

/*=====
hilltop *get_six18_alignment (char x[], char y[], int mode)
/*
 * Dynamic programming routine to match two nucleotide sequences, using
 * square memory, using a six-state node.
 * Mode = 0: one hilltop as output.
 * Mode = 1: separate at long gaps.
 */
{
    int i, j, best_i = 0, best_j = 0, best_s = DIAG;
    int tmp, id_count, sim_count, no_n_count;
    int lenx, leny;
    double max_score = -1000000.0;
    sixnode *diag, *left, *down, *current, *mat, zero;
    hilltop *res, *ht;

    create_hilltop (&ht);
    lenx = strlen (x);
    leny = strlen (y);

    if ((lenx*leny*sizeof(sixnode) > MAX_FULL*MAX_FULL*sizeof(node)) {
        /* We need a linear memory alignment here (which is slower) */
        call_linear_six18_alignment (x, y, ht);
        if (ht->x[0] == '-' || ht->x0 != 1;
            if (ht->y[0] == '-' || ht->y0 != 1;
            ht->final = 1;
            improve_six18_result (ht, mode);
            return (ht);
        }

        /* allocate memory and initialize matrix boundaries */
        if (! (mat = (sixnode *) malloc ((1+lenx*leny * sizeof (sixnode))))
            error(1, -1, "couldn't allocate memory for nodes 1 (%d %d).\n",
                lenx, leny);

        zero.mat = zero.ins = zero.del = zero.xgap = zero.ygap = zero.alt = 0;

        /* main loop */
        for (i = 0; i < lenx; i++) {
            /* Advance one column and initialize pointers */

```

dp.c

```

current = &mat[i*leny];
if (i == 0) left = &zero;
else left = &mat[(i-1)*leny];
diag = &zero; down = &zero;

for (j = 0; j < leny; j++) {
    current->match = diag->match;
    if (diag->ins > current->match) current->match = diag->ins;
    if (diag->del > current->match) current->match = diag->del;
    if (diag->xgap-XL_GO > current->match) current->match = diag->xgap-XL_GO;
    if (diag->ygap-YL_GO > current->match) current->match = diag->ygap-YL_GO;
    if (diag->alt-XL_GO > current->match) current->match = diag->alt-XL_GO;
    current->match += exact_sc[x[i]-'A'][y[j]-'A'];

    current->ins = left->match - X_GO;
    if (left->ins - X_GE > current->ins) current->ins = left->ins - X_GE;

    current->del = down->match - Y_GO;
    if (down->del - Y_GE > current->del) current->del = down->del - Y_GE;

    current->xgap = left->match - XL_GO;
    if (left->xgap - XL_GE > current->xgap) current->xgap = left->xgap - XL_GE;
    if (left->alt - XL_GE > current->xgap) current->xgap = left->alt - XL_GE;

    current->ygap = down->match - YL_GO;
    if (down->ygap - YL_GE > current->ygap) current->ygap = down->ygap - YL_GE;
    if (down->alt - YL_GE > current->ygap) current->ygap = down->alt - YL_GE;

    current->alt = diag->match - YL_GO;
    if (diag->alt - YL_GE > current->alt) current->alt = diag->alt - YL_GE;

    /* No gap open penalty for first gap (if at edge) */
    if (i == 0 || j == 0)
        current->xgap = current->ygap = current->alt = 0;

    /* Save best state and node */
    if (i == lenx-1 || j == leny-1) {
        if (current->match >= max_score) {
            max_score = current->match;
            best_i = i; best_j = j; best_s = DIAG;
        }
        if (current->ins >= max_score) {
            max_score = current->ins;
            best_i = i; best_j = j; best_s = LEFT;
        }
        if (current->del >= max_score) {
            max_score = current->del;
            best_i = i; best_j = j; best_s = DOWN;
        }
        if (current->xgap >= max_score) {
            max_score = current->xgap;
            best_i = i; best_j = j; best_s = XGAP;
        }
        if (current->ygap >= max_score) {
            max_score = current->ygap;
            best_i = i; best_j = j; best_s = YGAP;
        }
        if (current->alt >= max_score) {
            max_score = current->alt;
            best_i = i; best_j = j; best_s = ALT;
        }
    }
}

```

```

)
}
/* Advance pointers for next node */
if (i > 0) {diag = left; left++; }
down = current; current++;
}

get_six18_match (x, y, mat, best_i, best_j, best_s, ht);
ht->score = max_score;
improve_six18_result (ht, mode);
free (mat);
return (ht);
}

/*=====
static calc_score(hilltop *ht)
/*
* (written by avner)
* given the alignment, calculate the score
*/

int i;
ht->score=0;
/* printf("x = %s\n" = %s\n", ht->x, ht->y); */
for (i=0; i < ht->len; i++) {
    /* printf("%2f, ", ht->score); */
    if (ht->x[i]!='-')
        {
            ht->score += X_GE;
            if (i==0 || ht->x[i-1]!='-')
                ht->score += X_GO/*-X_GE*/;
        }
    else if (ht->y[i]!='-')
        {
            ht->score += X_GE;
            if (i==0 || ht->y[i-1]!='-')
                ht->score += X_GO/*-X_GE*/;
        }
    else ht->score += exact_sc[ht->x[i]-'A'][ht->y[i]-'A'];
}

/*=====
void improve_six18_result (hilltop *ht, int mode)
/*
int xpos, ypos, i, first;
hilltop *ht2;

if (mode == 0) {
    for (i = 0; i < strlen (ht->diff); i++)
        if (ht->diff[i] == '-') ht->diff[i] = ' ';
    } else {
        xpos = ht->xt /* + 1 */;
        ypos = ht->yt /* + 1 */;
        for (i = strlen (ht->diff)-1, first=1; i >= 0; i--, first=0) {
            if (ht->diff[i] == '-') {
                if (i!=first)
                    {
                        create_hilltop (&(ht2));
                    }
            }
        }
    }
}

```

dp.c

d/p.c


```

nice_x[loc] = x[i];
nice_y[loc] = y[i];
nice_diff[loc] = '=';
if (i == 0 || j == 0) last = NONE;
else {
    tmp = current->ygap;
    if (fabs(left->ygap - tmp-XL_GE) < EPSILON) last = XGAP;
    else if (fabs(left->alt - tmp-XL_GE) < EPSILON) last = ALT;
    else if (fabs(left->match - tmp-XL_GO) < EPSILON) last = DIAG;
    else printf ("error in back-tracking 4, %d %d\n", i, j);
} if (last == NONE) break;
i--;
} else if (last == YGAP) {
    nice_x[loc] = '-';
    nice_y[loc] = y[j];
    nice_diff[loc] = '=';
    if (i == 0 || j == 0) last = NONE;
    else {
        tmp = current->ygap;
        if (fabs(down->ygap - tmp-YL_GE) < EPSILON) last = YGAP;
        else if (fabs(down->alt - tmp-YL_GE) < EPSILON) last = ALT;
        else if (fabs(down->match - tmp-YL_GO) < EPSILON) last = DIAG;
        else printf ("error in back-tracking 5, %d %d\n", i, j);
    } if (last == NONE) break;
j--;
} else if (last == ALT) {
    nice_x[loc] = x[i];
    nice_y[loc] = y[j];
    nice_diff[loc] = '=';
    if (i == 0 || j == 0) last = NONE;
    else {
        tmp = current->alt;
        if (fabs(diag->alt - tmp-YL_GE) < EPSILON) last = ALT;
        else if (fabs(diag->match - tmp-YL_GO) < EPSILON) last = DIAG;
        else printf ("error in back-tracking 6, %d %d\n", i, j);
    } if (last == NONE) break;
i--; j--;
}
}

ht->final = 1;
ht->len = strlen (nice_y[loc]);
ht->x = subseq (nice_x[loc], 0, ht->len);
ht->y = subseq (nice_y[loc], 0, ht->len);
ht->diff = subseq (nice_diff[loc], 0, ht->len);
ht->xt = it;
ht->yt = jt;
ht->x0 = i; if (ht->x[0] == '-') ht->x0 += 1;
ht->y0 = j; if (ht->y[0] == '-') ht->y0 += 1;
free (nice_x);
free (nice_y);
free (nice_diff);

id_count = sim_count = no_n_count = 0;
for (i = 0; i < strlen (ht->diff); i++) {
    id_count += (ht->diff[i] == '|');
    if (ht->x[i] != '-' && ht->y[i] != '-')
        sim_count += (exact_sc[ht->x[i]-'A'][ht->y[i]-'A'] > 0);
    if (ht->x[i] != 'N' && ht->y[i] != 'N') no_n_count++;
}

```

```

ht->id_percent = 100*(double) (id_count) / (double) (no_n_count);
ht->sim_percent = 100*(double) (sim_count) / (double) (no_n_count);
}

/*=====*/
void call_linear_six18_alignment (char x[], char y[], hilltop *ht)
/*
 * Dynamic programming routine for first stage of matching two nucleotide
 * sequences, using linear memory, in the 6-state, 18-transition model.
 */
{
    int i, j, k, x0, y0, s0, xt, yt, st;
    int lenx, leny, loop_end, h_line, v_line, update;
    lsixnode *diag, *left, *down, *current, *tmp_node;
    lsixnode *col, *prevcol, zero, bad;
    hilltop *bottom, *top;
    double max_score = -1000000.0;

    lenx = strlen (x);
    leny = strlen (y);

    /* allocate memory */
    if (!col = (lsixnode *) malloc ((1+leny * sizeof (lsixnode))))
        error(1, -1, "couldn't allocate memory for nodes in linear alignment.\n");
    if (!prevcol = (lsixnode *) malloc ((1+leny * sizeof (lsixnode))))
        error(1, -1, "couldn't allocate memory for nodes in linear alignment.\n");

    zero.sc[0] = zero.sc[1] = zero.sc[2] = zero.sc[3] = zero.sc[4] = zero.sc[5] = 0.0;
    bad.sc[0] = bad.sc[1] = bad.sc[2] = bad.sc[3] = bad.sc[4] = bad.sc[5] = -1000000.0;
    for (j = 0; j < leny; j++) col[j] = prevcol[j] = zero;

    /* main loop */
    for (i = 0; i < lenx; i++) {
        /* Advance one column and initialize pointers */
        tmp_node = col; col = prevcol; prevcol = tmp_node;
        current = col; left = prevcol;
        diag = &zero; down = &zero;

        for (j = 0; j < leny; j++) {
            /* state 0 : match */
            current->sc[0] = diag->sc[0];
            current->x0[0] = diag->x0[0];
            current->y0[0] = diag->y0[0];
            current->s0[0] = diag->s0[0];
            if (diag->sc[1] > current->sc[0]) {
                current->sc[0] = diag->sc[1];
                current->x0[0] = diag->x0[1];
                current->y0[0] = diag->y0[1];
                current->s0[0] = diag->s0[1];
            }
            if (diag->sc[2] > current->sc[0]) {
                current->sc[0] = diag->sc[2];
                current->x0[0] = diag->x0[2];
                current->y0[0] = diag->y0[2];
                current->s0[0] = diag->s0[2];
            }
            if (diag->sc[3] - XL_GO > current->sc[0]) {
                current->sc[0] = diag->sc[3] - XL_GO;
                current->x0[0] = diag->x0[3];
            }
        }
    }
}

```



```

current->y0[0] = diag->y0[3];
current->s0[0] = diag->s0[3];
}
if (diag->sc[4] - XL_GO > current->sc[0]) {
    current->sc[0] = diag->sc[4] - XL_GO;
    current->x0[0] = diag->x0[4];
    current->y0[0] = diag->y0[4];
    current->s0[0] = diag->s0[4];
}
if (diag->sc[5] - XL_GO > current->sc[0]) {
    current->sc[0] = diag->sc[5] - XL_GO;
    current->x0[0] = diag->x0[5];
    current->y0[0] = diag->y0[5];
    current->s0[0] = diag->s0[5];
}
current->sc[0] += exact_sc[x[i+ht->x0] - 'A'] [y[j+ht->y0] - 'A'];

/* State 1: insert */
current->sc[1] = left->sc[0] - X_GO;
current->x0[1] = left->x0[0];
current->y0[1] = left->y0[0];
current->s0[1] = left->s0[0];
if (left->sc[1] - X_GE > current->sc[1]) {
    current->sc[1] = left->sc[1] - X_GE;
    current->x0[1] = left->x0[1];
    current->y0[1] = left->y0[1];
    current->s0[1] = left->s0[1];
}

/* State 2: delete */
current->sc[2] = down->sc[0] - Y_GO;
current->x0[2] = down->x0[0];
current->y0[2] = down->y0[0];
current->s0[2] = down->s0[0];
if (down->sc[2] - Y_GE > current->sc[2]) {
    current->sc[2] = down->sc[2] - Y_GE;
    current->x0[2] = down->x0[2];
    current->y0[2] = down->y0[2];
    current->s0[2] = down->s0[2];
}

/* state 3: X-gap */
current->sc[3] = left->sc[0] - XL_GO;
current->x0[3] = left->x0[0];
current->y0[3] = left->y0[0];
current->s0[3] = left->s0[0];
if (left->sc[3] - XL_GE > current->sc[3]) {
    current->sc[3] = left->sc[3] - XL_GE;
    current->x0[3] = left->x0[3];
    current->y0[3] = left->y0[3];
    current->s0[3] = left->s0[3];
}
if (left->sc[5] - XL_GE > current->sc[3]) {
    current->sc[3] = left->sc[5] - XL_GE;
    current->x0[3] = left->x0[5];
    current->y0[3] = left->y0[5];
    current->s0[3] = left->s0[5];
}

/* state 4: Y-gap */

```

dp.c

```

current->sc[4] = down->sc[0] - YL_GO;
current->x0[4] = down->x0[0];
current->y0[4] = down->y0[0];
current->s0[4] = down->s0[0];
if (down->sc[4] - YL_GE > current->sc[4]) {
    current->sc[4] = down->sc[4] - YL_GE;
    current->x0[4] = down->x0[4];
    current->y0[4] = down->y0[4];
    current->s0[4] = down->s0[4];
}
if (down->sc[5] - YL_GE > current->sc[4]) {
    current->sc[4] = down->sc[5] - YL_GE;
    current->x0[4] = down->x0[5];
    current->y0[4] = down->y0[5];
    current->s0[4] = down->s0[5];
}

/* State 5: Alternate */
current->sc[5] = diag->sc[0] - YL_GO;
current->x0[5] = diag->x0[0];
current->y0[5] = diag->y0[0];
current->s0[5] = diag->s0[0];
if (diag->sc[5] - YL_GE > current->sc[5]) {
    current->sc[5] = diag->sc[5] - YL_GE;
    current->x0[5] = diag->x0[5];
    current->y0[5] = diag->y0[5];
    current->s0[5] = diag->s0[5];
}

/* initialize x0, y0 and s0 */
if ((i == 0) || (j == 0)) {
    for (k = 0; k < 6; k++) {
        current->x0[k] = i;
        current->y0[k] = j;
        current->s0[k] = k;
    }
}

/* Save best end point */
if (i == lenx-1 || j == leny-1)
    for (k = 0; k < 6; k++)
        if (current->sc[k] >= max_score) {
            max_score = current->sc[k];
            x0 = current->x0[k];
            y0 = current->y0[k];
            s0 = current->s0[k];
            xt = i;
            yt = j;
            st = k;
        }

/* Advance pointers for next node */
diag = left; left++; down = current; current++;
}
free (col);
free (prevcol);
ht->x0 = x0;

```

dp.c

A-231

```

ht->y0 = y0;
ht->xt = xt;
ht->yt = yt;
linear_six18_alignment (x, y, ht, s0, st);
}

/* ===== */
void linear_six18_alignment (char x[], char y[], hilltop *ht, int bs, int es)
/*
 * Recursive dynamic programming routine to match two nucleotide sequences,
 * using linear memory, in the 6-state, 18-transition model.
 */
{
    int i, j, k, x0, y0, s0;
    int lenx, leny, loopend, h_line, v_line, update;
    lsixnode *diag, *left, *down, *current, *tmp_node;
    lsixnode *col, *prevcol, *zero, *bad;
    hilltop *bottom, *top;

    /* ===== */
    /* In linear alignment x0=xd, xt=xd, y0=yd, yt=yd, bs=bd, es=ed\n",
       ht->x0, ht->xt, ht->y0, ht->yt, bs, es); */

    lenx = ht->xt - ht->x0 + 1;
    leny = ht->yt - ht->y0 + 1;
    /* First check if this is the end of the recursion */
    if (lenx <= 2 && leny <= 2) {
        if (! (ht->x = (char *) malloc (3)))
            error(1, -1, "couldn't allocate memory for alignment string.\n");
        if (! (ht->y = (char *) malloc (3)))
            error(1, -1, "couldn't allocate memory for alignment string.\n");
        if (! (ht->diff = (char *) malloc (3)))
            error(1, -1, "couldn't allocate memory for alignment string.\n");
        ht->x[2] = ht->y[2] = ht->diff[2] = '\0';
        ht->diff[0] = ht->diff[1] = ' ';
        ht->x[0] = x[ht->x0]; ht->y[0] = y[ht->y0];
        ht->x[1] = x[ht->xt]; ht->y[1] = y[ht->yt];
        if (bs == 0) {
            if (exact_sc[ht->x[0] - 'A'][ht->y[0] - 'A'] > 0) ht->diff[0] = ' ';
            if (ht->x[0] == ht->y[0]) ht->diff[0] = '|';
        } else if (bs == 1)
            ht->y[0] = '-';
        else if (bs == 2)
            ht->x[0] = '-';
        else if (bs == 3) {
            ht->y[0] = '-';
            ht->diff[0] = '=';
        } else if (bs == 4) {
            ht->x[0] = '-';
            ht->diff[0] = '=';
        } else
            ht->diff[0] = '=';

        if (es == 0) {
            if (exact_sc[ht->x[1] - 'A'][ht->y[1] - 'A'] > 0) ht->diff[1] = ' ';
            if (ht->x[1] == ht->y[1]) ht->diff[1] = '|';
        } else if (es == 1)
            ht->y[1] = '-';
        else if (es == 2)

```

```

ht->x[1] = '-';
else if (es == 3) {
    ht->y[1] = '-';
    ht->diff[1] = '=';
} else if (es == 4) {
    ht->x[1] = '-';
    ht->diff[1] = '=';
} else
    ht->diff[1] = '-';
ht->len = 2;
return;
}

/* Define h_line and v_line */
h_line = (lenx-1) / 2; if (lenx == 1) h_line = -1;
v_line = (leny-1) / 2; if (leny == 1) v_line = -1;
/* allocate memory */
if (! (col = (lsixnode *) malloc ((1+leny * sizeof (lsixnode))))
    error(1, -1, "couldn't allocate memory for nodes in linear alignment.\n");
if (! (prevcol = (lsixnode *) malloc ((1+leny * sizeof (lsixnode))))
    error(1, -1, "couldn't allocate memory for nodes in linear alignment.\n");
zero.sc[0] = zero.sc[1] = zero.sc[2] = zero.sc[3] = zero.sc[4] = zero.sc[5] = 0.0;
bad.sc[0] = bad.sc[1] = bad.sc[2] = bad.sc[3] = bad.sc[4] = bad.sc[5] = -1000000;
for (j = 0; j < leny; j++) col[j] = prevcol[j] = bad;

/* main loop */
for (i = 0; i < lenx; i++) {
    /* Advance one column and initialize pointers */
    tmp_node = col; col = prevcol; prevcol = tmp_node;
    current = col; left = prevcol;
    diag = &bad; down = &bad;
    if (i == 0) diag = &zero;
    for (j = 0; j < leny; j++) {
        /* initialize x0, y0 and s0 */
        if ((i == h_line && j >= v_line) || (i >= h_line && j == v_line)) {
            for (k = 0; k < 6; k++) {
                current->x0[k] = i;
                current->y0[k] = j;
                current->s0[k] = k;
            }
        }
        /* update the scores */
        update = 0;
        if (i > h_line && j > v_line) update = 1;

        /* state 0 : match */
        current->sc[0] = diag->sc[0];
        if (update) {
            current->x0[0] = diag->x0[0];
            current->y0[0] = diag->y0[0];
            current->s0[0] = diag->s0[0];
        }
        if (diag->sc[1] > current->sc[0]) {
            current->sc[0] = diag->sc[1];
            if (update) {
                current->x0[0] = diag->x0[1];
                current->y0[0] = diag->y0[1];
                current->s0[0] = diag->s0[1];
            }
        }
    }
}

```

```

        current->s0[0] = diag->s0[1];
    }
}
if (diag->sc[2] > current->sc[0]) {
    current->sc[0] = diag->sc[2];
    if (update) {
        current->x0[0] = diag->x0[2];
        current->y0[0] = diag->y0[2];
        current->s0[0] = diag->s0[2];
    }
}
if (diag->sc[3] - XL_GO > current->sc[0]) {
    current->sc[0] = diag->sc[3] - XL_GO;
    if (update) {
        current->x0[0] = diag->x0[3];
        current->y0[0] = diag->y0[3];
        current->s0[0] = diag->s0[3];
    }
}
if (diag->sc[4] - XL_GO > current->sc[0]) {
    current->sc[0] = diag->sc[4] - XL_GO;
    if (update) {
        current->x0[0] = diag->x0[4];
        current->y0[0] = diag->y0[4];
        current->s0[0] = diag->s0[4];
    }
}
if (diag->sc[5] - XL_GO > current->sc[0]) {
    current->sc[0] = diag->sc[5] - XL_GO;
    if (update) {
        current->x0[0] = diag->x0[5];
        current->y0[0] = diag->y0[5];
        current->s0[0] = diag->s0[5];
    }
}
current->sc[0] += exact_sc[x[i+ht->x0] - 'A'] [y[j+ht->y0] - 'A'];

/* State 1: insert */
current->sc[1] = left->sc[0] - X_GO;
if (update) {
    current->x0[1] = left->x0[0];
    current->y0[1] = left->y0[0];
    current->s0[1] = left->s0[0];
}
if (left->sc[1] - X_GE > current->sc[1]) {
    current->sc[1] = left->sc[1] - X_GE;
    if (update) {
        current->x0[1] = left->x0[1];
        current->y0[1] = left->y0[1];
        current->s0[1] = left->s0[1];
    }
}

/* State 2: delete */
current->sc[2] = down->sc[0] - Y_GO;
if (update) {
    current->x0[2] = down->x0[0];
    current->y0[2] = down->y0[0];
    current->s0[2] = down->s0[0];
}

```

dp.c

```

if (down->sc[2] - Y_GE > current->sc[2]) {
    current->sc[2] = down->sc[2] - Y_GE;
    if (update) {
        current->x0[2] = down->x0[2];
        current->y0[2] = down->y0[2];
        current->s0[2] = down->s0[2];
    }
}

/* state 3: X-gap */
current->sc[3] = left->sc[0] - XL_GO;
if (update) {
    current->x0[3] = left->x0[0];
    current->y0[3] = left->y0[0];
    current->s0[3] = left->s0[0];
}
if (left->sc[3] - XL_GE > current->sc[3]) {
    current->sc[3] = left->sc[3] - XL_GE;
    if (update) {
        current->x0[3] = left->x0[3];
        current->y0[3] = left->y0[3];
        current->s0[3] = left->s0[3];
    }
}
if (left->sc[5] - XL_GE > current->sc[3]) {
    current->sc[3] = left->sc[5] - XL_GE;
    if (update) {
        current->x0[3] = left->x0[5];
        current->y0[3] = left->y0[5];
        current->s0[3] = left->s0[5];
    }
}

/* state 4: Y-gap */
current->sc[4] = down->sc[0] - YL_GO;
if (update) {
    current->x0[4] = down->x0[0];
    current->y0[4] = down->y0[0];
    current->s0[4] = down->s0[0];
}
if (down->sc[4] - YL_GE > current->sc[4]) {
    current->sc[4] = down->sc[4] - YL_GE;
    if (update) {
        current->x0[4] = down->x0[4];
        current->y0[4] = down->y0[4];
        current->s0[4] = down->s0[4];
    }
}
if (down->sc[5] - YL_GE > current->sc[4]) {
    current->sc[4] = down->sc[5] - YL_GE;
    if (update) {
        current->x0[4] = down->x0[5];
        current->y0[4] = down->y0[5];
        current->s0[4] = down->s0[5];
    }
}

/* State 5: Alternate */
current->sc[5] = diag->sc[0] - YL_GO;
if (update) {

```

dp.c

```

current->x0[5] = diag->x0[0];
current->y0[5] = diag->y0[0];
current->s0[5] = diag->s0[0];
}
if (diag->sc[5] - YL_GE > current->sc[5]) {
    if (update) {
        current->x0[5] = diag->x0[5];
        current->y0[5] = diag->y0[5];
        current->s0[5] = diag->s0[5];
    }
}

if (i == 0 && j == 0)
    for (k = 0; k < 6; k++)
        if (bs == k) current->sc[k] = zero.sc[k];
        else current->sc[k] = bad.sc[k];
/* Advance pointers for next node */
diag = left; left++; down = current; current++;
}

x0 = down->x0[es] + ht->x0;
y0 = down->y0[es] + ht->y0;
s0 = down->s0[es];

create_hilltop (&bottom);
bottom->type = top->type = ht->type;
bottom->x0 = ht->x0; bottom->y0 = ht->y0;
bottom->xt = x0; bottom->yt = y0;
top->x0 = x0; top->y0 = y0;
top->xt = ht->xt; top->yt = ht->yt;
linear_six18_alignment (x, y, bottom, bs, s0);
linear_six18_alignment (x, y, top, s0, es);
ht->len = bottom->len + top->len - 1;

if (! (ht->x = (char *) malloc (ht->len+1)))
    error(1, -1, "couldn't allocate memory for alignment string.\n");
if (! (ht->y = (char *) malloc (ht->len+1)))
    error(1, -1, "couldn't allocate memory for alignment string.\n");
if (! (ht->diff = (char *) malloc (ht->len+1)))
    error(1, -1, "couldn't allocate memory for alignment string.\n");

for (i = 0; i < bottom->len; i++) {
    ht->x[i] = bottom->x[i];
    ht->y[i] = bottom->y[i];
    ht->diff[i] = bottom->diff[i];
}
for (i = 0; i < top->len; i++) {
    ht->x[i+bottom->len-1] = top->x[i];
    ht->y[i+bottom->len-1] = top->y[i];
    ht->diff[i+bottom->len-1] = top->diff[i];
}
ht->x[ht->len] = ht->y[ht->len] = ht->diff[ht->len] = '\0';
ht->final = 1;

destroy_hilltop (&bottom);
destroy_hilltop (&top);
free (col);
free (prevcol);

```

```

)
/*=====
hilltop *band_sw (char x[], int x0, int y0, int width)
/*
 * Dynamic programming routine. Returns the best overlap alignment between
 * it's two input sequences in a band.
 */
{
    int i, j, best_i = 0, best_j = 0, tmp, id_count, sim_count, no_n_count;
    int lenx, leny, mode, loop_end, band, real_j;
    double max_score = -100000.0;
    node *diag, *left, *down, *current, *mat, zero, bad;
    hilltop *ht;

    create_hilltop (&ht);
    lenx = strlen (x);
    leny = strlen (y);
    if (lenx == 0 || leny == 0)
        error (1, -1, "Sequence of length zero in band_sw\n");
    band = 2*width + 1;

    /* allocate memory and initialize matrix boundaries */
    if (! (mat = (node *) malloc ((lenx*band * sizeof (node))))
        error(1, -1, "couldn't allocate memory for band nodes 1 (%d %d).\n",
            lenx, leny);

    zero.match = zero.ins = zero.del = 0;
    bad.match = bad.ins = bad.del = -100000;

    /* main loop */
    for (i = 0; i < lenx; i++) {
        for (real_j = 0; real_j < band; real_j++) {
            j = real_j + i - x0 + y0 - width;
            if (j >= leny || j < 0) continue;

            /* Set pointers */
            current = &mat[i*band + real_j];
            if (i == 0 || j == 0) diag = &zero;
            else diag = &mat[(i-1)*band + real_j];
            if (i == 0) left = &zero;
            else if (real_j == band - 1) left = &bad;
            else left = &mat[(i-1)*band + real_j+1];
            if (j == 0) down = &zero;
            else if (real_j == 0) down = &bad;
            else down = &mat[i*band + real_j-1];

            /* Calculate node */
            current->match = diag->match;
            if (diag->ins > current->match) current->match = diag->ins;
            if (diag->del > current->match) current->match = diag->del;
            current->match += exact_sc[x[i]-'A'][y[j]-'A'];

            current->ins = left->match - X_GO;
            if (left->ins - X_GE > current->ins) current->ins = left->ins - X_GE;
            current->del = down->match - Y_GO;

```

```

if (down->del - Y_GE > current->del) current->del = down->del - Y_GE;

/* Save best node */
if (i == lenx-1 || j == leny-1)
    if ((current->match) >= max_score) {
        max_score = current->match;
        best_i = i;
        best_j = j;
    }
}

get_band_match (x, y, x0, y0, width, mat, best_i, best_j, ht);
free (mat);
return (ht);
}

/* Extract the best solution */
{
    char *nice_x, *nice_y, *nice_diff;
    int last, i, j, loc, end, id_count, sim_count, no_n_count, lenx, leny;
    int real_j, band;
    double tmp;
    node *current, *diag, *left, *down, *bad;

    lenx = strlen (x);
    leny = strlen (y);
    bad_match = bad.ins = bad.del = -100000;
    band = 2*width + 1;

    end = it + jt;
    if (((nice_x = (char *)malloc (end+2)) == NULL) ||
        ((nice_y = (char *)malloc (end+2)) == NULL) ||
        ((nice_diff = (char *)malloc (end+2)) == NULL))
        error (1, "-1, \"Can't allocate memory for nice_strings\");
    nice_x[end+1] = nice_y[end+1] = nice_diff[end+1] = '\0';

    i = it; j = jt;
    last = DIAG;

    for (loc = end; last != NONE; loc--) {
        real_j = j - i + x0 - y0 + width;

        current = &mat[i * band + real_j];
        if (real_j < band-1)
            left = &mat[(i-1)*band + real_j+1];
        else left = &bad;
        if (real_j > 0)
            down = &mat[i * band + real_j-1];
        else down = &bad;
        diag = &mat[(i-1)*band + real_j];

        if (last == DIAG) {
            nice_x[loc] = x[i];
            nice_y[loc] = y[j];
            if (x[i] == y[j]) nice_diff[loc] = '|';
            else nice_diff[loc] = (exact_sc[x[i]-'A'][y[j]-'A'] > 0) ? ':' : ' ';
            tmp = current->match - exact_sc[x[i]-'A'][y[j]-'A'];

```

dp.c

```

if (i == 1 || j == 1) last = NONE;
else {
    if (fabs(diag->match - tmp) < EPSILON) last = DIAG;
    else if (fabs(diag->del - tmp) < EPSILON) last = DOWN;
    else if (fabs(diag->ins - tmp) < EPSILON) last = LEFT;
    else
        printf ("error in band back-tracking 1, %d %d\n", i, j);
} if (last == NONE) break;
i--; j--;
} else if (last == LEFT) {
    nice_x[loc] = x[i];
    nice_y[loc] = y[j];
    nice_diff[loc] = '-';
    if (i == 1 || j == 1) last = NONE;
    else {
        tmp = current->ins;
        if (fabs(left->match - tmp-X_GO) < EPSILON) last = DIAG;
        else if (fabs(left->ins - tmp-X_GE) < EPSILON) last = LEFT;
        else printf ("error in band back-tracking 2, %d %d\n", i, j);
    } i--;
} else if (last == DOWN) {
    nice_x[loc] = x[i];
    nice_y[loc] = y[j];
    nice_diff[loc] = '.';
    if (i == 1 || j == 1) last = NONE;
    else {
        tmp = current->del;
        if (fabs(down->del - tmp-Y_GO) < EPSILON) last = DOWN;
        else if (fabs(down->match - tmp-Y_GE) < EPSILON) last = DIAG;
        else printf ("error in band back-tracking 3, %d %d\n", i, j);
    } j--;
}
}

ht->final = 1;
ht->len = strlen (nice_y[loc]);
ht->x = subseq (nice_x[loc], 0, ht->len);
ht->y = subseq (nice_y[loc], 0, ht->len);
ht->diff = subseq (nice_diff[loc], 0, ht->len);
ht->xt = it;
ht->yt = jt;
ht->x0 = i;
ht->y0 = j;
destroy_hilltop (&ht);
free (nice_x);
free (nice_y);
free (nice_diff);
}

/*=====
char uppercase (char c)
{
    if (c >= 97 && c <= 122)
        return (c - 97 + 65);
    else return (c);
}

/*=====
char lowercase (char c)
{

```

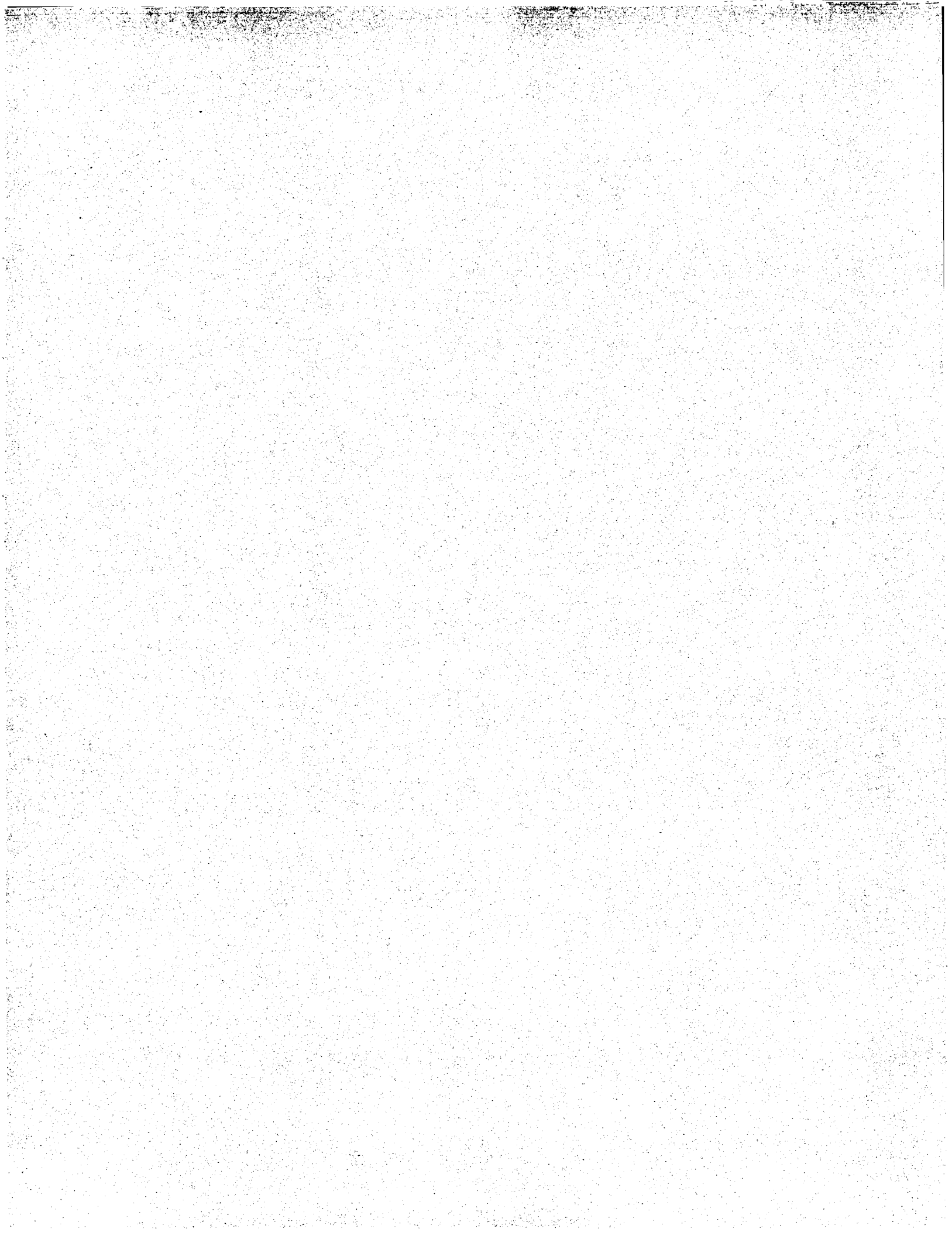
dp.c

Sun Aug 9 10:39:55 1998

Listing for Adam Sartiell

```
if (c >= 65 && c <= 90)
    return (c + 97 - 65);
else return (c);
)
```

dp:c



```

/*
 * Copyright 1996 Compugen, Ltd.
 * Authorization to use this code is given solely to Compugen customers.
 * This authorization is limited by the terms of the Compugen Hardware and
 * Software License Agreement.
 *
 * Last update: $Date: 1998/04/12 15:00:16 $ by $Author: prod $
 * Revision: $Revision: 1.1 $
 */
#include <stdio.h>
#include <strings.h>
#include <sys/types.h>
#include <malloc.h>
#include <math.h>

#include "hilltops.h"

/*=====*/
char *subseq (char *seq, int first, int last)
/*
 * Cut a sub-sequence from a given input sequence. If first > last, flip it
 * and translate C<->G, A<->T, (and subsets as well).
 */
{
    char *res;
    int i, len, newlen, flip = 0;

    len = strlen (seq);

    if (first < 0) first = 0;
    if (first >= len) first = len-1;
    if (last < 0) last = 0;
    if (last >= len) last = len-1;

    newlen = last - first;
    if (newlen < 0) {
        flip = 1;
        newlen = -newlen;
    }
    newlen++;
    if (! (res = (char *) malloc (newlen+1)))
        error(1, -1, "subseq: can't allocate string\n");

    res[newlen] = '\0';
    if (flip)
        for (i = 0; i < newlen; i++)
            res[i] = inv(seq[first-i]);
    else
        for (i = 0; i < newlen; i++)
            res[i] = seq[first+i];

    return (res);
}

/*=====*/
char inv (char c)
/*
 * return base pair
 */

```

hilltops.c

```

{
    char alphabet[] = "-ACMGRSVTWYHKDBN";
    /* K = G or T M = A or C R = G or A S = G or C W = A or T Y = T or C
     * B = G or T or C D = G or A or T H = A or C or T V = G or C or A
     * N = A or C or G or T */
    if (c == 'A') return 'T';
    if (c == 'C') return 'G';
    if (c == 'G') return 'C';
    if (c == 'T') return 'A';
    if (c == '-') return '-';
    if (c == 'N') return 'N';
    if (c == 'R') return 'Y';
    if (c == 'Y') return 'R';
    if (c == 'K') return 'M';
    if (c == 'M') return 'K';
    if (c == 'S') return 'S';
    if (c == 'W') return 'W';
    if (c == 'B') return 'V';
    if (c == 'D') return 'H';
    if (c == 'H') return 'D';
    if (c == 'V') return 'B';
    if (c == 'X') return 'X';
    return c;
}

/*=====*/
void create_segment (segment **obj)
{
    if (! ((*obj) = (segment *) malloc (sizeof(segment))))
        error(1, -1, "couldn't allocate memory for segment.\n");

    (*obj)->next = NULL;
    (*obj)->start = 0;
    (*obj)->end = 0;
    (*obj)->score = 0.0;
    (*obj)->best_y = 0;
    (*obj)->x0 = 0;
    (*obj)->y0 = 0;
}

/*=====*/
void destroy_segment (segment **obj)
{
    free (*obj);
    *obj = NULL;
}

/*=====*/
void destroy_segment_list (segment **obj)
{
    segment *segg1, *segg2;
    for (segg1 = *obj; segg1 != NULL; ) {
        segg2 = segg1->next;
        destroy_segment (&segg1);
        segg1 = segg2;
    }
    *obj = NULL;
}

/*=====*/

```

hilltops.c


```

void copy_segment (segment from, segment *obj)
{
    obj->next = from.next;
    obj->start = from.start;
    obj->end = from.end;
    obj->score = from.score;
    obj->best_y = from.best_y;
    obj->x0 = from.x0;
    obj->y0 = from.y0;
}
/*=====*/
void create_hilltop (hilltop **obj)
/*
 * Initialize a sequence alignment structure
 */
{
    if (!(*obj) = (hilltop *) malloc (sizeof(hilltop)))
        error(1, -1, "couldn't allocate memory for hilltop.\n");
    (*obj)->next = NULL;
    (*obj)->type = UNDEFINED;
    (*obj)->final = 0;
    (*obj)->len = 0;
    (*obj)->x0 = 0;
    (*obj)->y0 = 0;
    (*obj)->xt = 0;
    (*obj)->yt = 0;
    (*obj)->score = 0;
    (*obj)->area = 0;
    (*obj)->id_percent = 0.0;
    (*obj)->sim_percent = 0.0;
    (*obj)->name[0] = '\0';
    (*obj)->boundary = NULL;
    (*obj)->x = NULL;
    (*obj)->y = NULL;
    (*obj)->diff = NULL;
}
/*=====*/
void destroy_hilltop (hilltop **obj)
{
    segment *seggp, *seggp2;
    if ((*obj)->x != NULL) free ((*obj)->x);
    if ((*obj)->y != NULL) free ((*obj)->y);
    if ((*obj)->diff != NULL) free ((*obj)->diff);
    destroy_segment_list (&((*obj)->boundary));
    free (*obj);
    *obj = NULL;
}
/*=====*/
void destroy_hilltop_list (hilltop **obj)
{
    hilltop *htp1, *htp2;
    for (htp1 = *obj; htp1 != NULL; ) {
        htp2 = htp1->next;

```

hilltops.c

```

destroy_hilltop (&htp1);
htp1 = htp2;
}
*obj = NULL;
}
/*=====*/
void copy_hilltop (hilltop from, hilltop *obj)
/*
 * Copy a sequence alignment structure
 */
{
    segment *seggp, **seggp;
    obj->next = from.next;
    obj->type = from.type;
    obj->final = from.final;
    obj->len = from.len;
    obj->x0 = from.x0;
    obj->y0 = from.y0;
    obj->xt = from.xt;
    obj->yt = from.yt;
    obj->score = from.score;
    obj->area = from.area;
    obj->id_percent = from.id_percent;
    obj->sim_percent = from.sim_percent;
    strcpy (obj->name, from.name);
    if (!obj->x = (char *) realloc (obj->x, 1+from.len))
        error(1, -1, "couldn't allocate memory for hilltop.\n");
    if (from.x) strcpy (obj->x, from.x);
    if (!obj->y = (char *) realloc (obj->y, 1+from.len))
        error(1, -1, "couldn't allocate memory for hilltop.\n");
    if (from.y) strcpy (obj->y, from.y);
    if (!obj->diff = (char *) realloc (obj->diff, 1+from.len))
        error(1, -1, "couldn't allocate memory for hilltop.\n");
    if (from.diff) strcpy (obj->diff, from.diff);
    seggp = &(obj->boundary);
    for (seggp = from.boundary; seggp != NULL; seggp = seggp->next) {
        create_segment (seggp);
        copy_segment (*seggp, *seggp);
        seggp = &((*seggp)->next);
    }
}
/*=====*/
void update_all_hilltops (segment *seg_list, hilltop **ht_list, int x, int w)
/*
 * Use the knowledge of the previous column and the segments of the
 * current column to update the area-map of all locations above cutoff.
 * "w" is used for wraparound. If not zero, hilltops wrap around the
 * horizontal edge of the matrix. In this case w should be equal to "leny".
 */
{
    int a, b, c, d, delete_flag;
    segment *seggp1, *seggp2, *old_boundary, **seggp;
    hilltop *htp, *htp2, **htpp;

```

hilltops.c

```

/* First, bring segments to correct form (separate x0 and y0) */
for (segpl = seg_list; segpl != NULL; segpl = segpl->next) {
    segpl->y0 = (segpl->x0 & 0xffff);
    segpl->x0 >= 16;
    for (; x - segpl->x0 > (1 << 16); segpl->x0 += (1 << 16));
    for (; segpl->best_y - segpl->y0 > (1 << 16); segpl->y0 += (1 << 16));
}

/* Go over the hilltops (which are updated to the previous column) */
for (htp = *ht_list; htp != NULL; htp = htp->next) {
    /* cut the boundary information out from the hilltop */
    old_boundary = htp->boundary;
    htp->boundary = NULL;

    /* In a hilltop, go over the segments it had in the previous column */
    for (segpl = old_boundary; segpl != NULL; segpl = segpl->next) {
        a = segpl->start;
        b = segpl->end;

        /* Now, try to find a segment in the new column which overlaps */
        for (seggp = kseg_list; *seggp != NULL; ) {
            c = (*seggp)->start;
            d = (*seggp)->end;
            if ((c <= b+1 && d >= a) || (w && (c == 0) && (b == w-1))) {
                /* Overlap : note the exact condition! */
                update_hilltop(htp, **seggp, x);
                /* Delete the (used) segment from the list */
                segp2 = (*seggp);
                (*seggp) = (*seggp)->next;
                destroy_segment(kseggp2);
            } else /* Only if we don't delete - advance */
                seggp = &((*seggp)->next);
        }
    }

    /* We now have the new hilltop boundary, unless it should be merged
    with a preceding hilltop. Check them all */
    for (segpl = old_boundary; segpl != NULL; segpl = segpl->next) {
        a = segpl->start;
        b = segpl->end;
        for (htpp = ht_list; *htpp != htp; ) {
            delete_flag = 0;
            for (seggp2 = (*htpp)->boundary; segp2 != NULL; segp2 = segp2->next) {
                c = segp2->start;
                d = segp2->end;
                if (c <= b+1 && d >= a) { /* Overlap : merge the hilltops */
                    merge_hilltops(htp, *htpp);
                    /* Delete the merged hilltop from the list */
                    htp2 = (*htpp);
                    (*htpp) = (*htpp)->next;
                    destroy_hilltop(htpp2);
                    delete_flag = 1;
                    break;
                }
            }
            /* Only if we don't delete - advance */
            if (!delete_flag) htp = &((*htpp)->next);
        }
    }
    destroy_segment_list(sold_boundary);
}

```

```

}

/* create new records for segments which are still unassociated */
for (segpl = seg_list; segpl != NULL; segpl = segpl->next) {
    create_hilltop(htpp);
    update_hilltop(htp, *segpl, x);
    htp->next = *ht_list;
    *ht_list = htp;
}
destroy_segment_list(kseg_list);
}

/*=====
void update_hilltop (hilltop *htp, segment seg, int x)
/*=====
/* We have ascertained that seg is associated with rec. We now have to
 * update the data in rec according to seg.
 */
{
    segment *seggp;

    htp->area += seg.end - seg.start + 1;
    if (htp->score < seg.score) {
        htp->score = seg.score;
        htp->xt = x;
        htp->yt = seg.best_y;
        htp->x0 = seg.x0;
        htp->y0 = seg.y0;
    }
    create_segment(kseggp);
    copy_segment(seg, seggp);
    seggp->next = htp->boundary;
    htp->boundary = seggp;
}

/*=====
void merge_hilltops (hilltop *htpl, hilltop *htp2)
/*=====
/* Merge the two areas into the first one.
 */
{
    segment *seggp;

    htp1->area += htp2->area;
    if (htpl->score < htp2->score) {
        htp1->score = htp2->score;
        htp1->xt = htp2->xt;
        htp1->yt = htp2->yt;
        htp1->x0 = htp2->x0;
        htp1->y0 = htp2->y0;
    }
    for (seggp = htp2->boundary; seggp->next != NULL; seggp = seggp->next);
    seggp->next = htp1->boundary;
    htp2->boundary = NULL;
}

/*=====

```



```

/*
 * Copyright 1996 Compugen, Ltd.
 * This authorization to use this code is given solely to Compugen customers.
 * This authorization is limited by the terms of the Compugen Hardware and
 * Software License Agreement.
 */

```

```

 * Last update: $Date: 1998/04/12 15:00:23 $ by $Author: prod $
 * Revision: $Revision: 1.1 $
 */

```

```

#include <stdio.h>
#include <strings.h>
#include <sys/types.h>
#include <malloc.h>
#include <math.h>

```

```

#include "dp.h"
#include "wdp.h"

```

```

/*definition of constants */

```

```

#define INF 100000

```

```

#define NONE -1
#define DIAG 0
#define LEFT 1
#define DOWN 2

```

```

/*typedefs */

```

```

typedef struct node {
    double match, ins, del;
} node;

```

```

typedef struct lnode {
    double sc[3];
    int x0[3], y0[3], s0[3];
} lnode;

```

```

/* local functions */
void get_wdp_match(char x[], char y[], node *mat,
    int it, int jt, int st, hilltop *ht);
void w_global_alignment(char in_x[], char in_y[], hilltop **ht);
void w_trim_hilltop(char in_x[], char in_y[], hilltop *ht, int cutoff);
void linear_wrap_alignment(char x[], char y[], hilltop *ht,
    int by, int bs, int ey, int es);
void continue_wrap_alignment(char in_x[], char in_y[], hilltop *ht,
    int by, int bs, int ey, int es);

```

```

/* static variables */
extern int X_GO, X_GE, Y_GO, Y_GE;
extern int sc[26][26]; /* work with a subset of the English Alphabet */
extern double exact_sc[26][26];

```

```

/*=====
void get_wrap_alignment(char in_x[], char in_y[], hilltop *ht)
/*=====

```

```

 * Wraparound Dynamic programming routine. matches a (long) nucleotide
 * sequence (x) with an unknown number of tandem duplications of another
 * (short) sequence (y), then finds alignment. Works in square memory.

```

```

/*
int i, j, best_i = 0, best_j = 0, tmp, lower;
int lenx, leny, max_score = 0; sim_count, id_count, no_n_count, loc;
node *diag, *left, *down, *current, *mat, zero;
char *x, *y;

```

```

/* copy input strings to working strings, shorten if necessary */
x = subseq(in_x, ht->xt); lenx = strlen(x);
y = subseq(in_y, 0, strlen(in_y)); leny = strlen(y);

```

```

/* If the dimensions are too big - do it the slow way */
if (lenx*leny > MAX_FULL*MAX_FULL) {
    free(ht->y);
    free(ht->x);

```

```

    free(ht->diff);
    ht->x0 += TANDEM_GAP;
    ht->xt -= TANDEM_GAP;
    linear_wrap_alignment(in_x, in_y, ht, -1, 0, -1, 0);
    ht->y0 = 0; ht->yt = leny - 1;
    ht->final = 1; ht->type = WRAPAROUND;
    ht->len = strlen(ht->x);

```

```

    id_count = sim_count = no_n_count = 0;
    for (i = 0; i < strlen(ht->diff); i++) {
        id_count += (ht->diff[i] == '|');
        if (ht->x[i] != '-' && ht->y[i] != '-')
            sim_count += (exact_sc[ht->x[i]-'A'][ht->y[i]-'A'] > 0);
        if (ht->x[i] != 'N' && ht->y[i] != 'N') no_n_count++;
    }
    ht->id_percent = 100*(double)(id_count) / (double)(no_n_count);
    ht->sim_percent = 100*(double)(sim_count) / (double)(no_n_count);

```

```

/* Change y, put upper/lower case letters in ht->y */
for (i = loc = lower = 0; i < ht->len; i++) {
    if (ht->y[i] != '-') {
        y[loc] = uppercase(ht->y[i]);
        if (lower) ht->y[i] = lowercase(ht->y[i]);
        else ht->y[i] = uppercase(ht->y[i]);
        loc++;
        if (loc == leny) { loc = 0; lower = 1 - lower; }
    }
}

```

```

strcpy(in_y, y); /* y string may have changed cyclically */
free(x);
free(y);
return;
}

```

```

/* allocate memory */
if (!mat = (node *) malloc((1+lenx*leny * sizeof (node))))
    error(1, -1, "Can't allocate memory for nodes.\n");

```

```

zero.match = zero.ins = zero.del = 0;
/* main loop */
for (i = 0; i < lenx; i++) {
    /* Advance one column and initialize pointers */
    if (i == 0) diag = left = &zero;
    else {
        diag = &mat[(i-1)*leny + leny-1]; left = &mat[(i-1)*leny];
    }
}

```

```

down = &zero; current = &mat[i*leny];

/* Loop over nodes in the column */
for (j = 0; j < leny; j++) {
    current->match = diag->match;
    if (diag->ins > current->match) current->match = diag->ins;
    if (diag->del > current->match) current->match = diag->del;
    current->match += exact_sc[x[i]-'A'][y[j]-'A'];
    if (current->match <= 0) current->match = 0;

    current->ins = left->match - X_GO;
    if (left->ins - X_GE > current->ins) current->ins = left->ins - X_GE;
    if (current->ins <= 0) current->ins = 0;

    current->del = down->match - Y_GO;
    if (down->del - Y_GE > current->del) current->del = down->del - Y_GE;
    if (current->del <= 0) current->del = 0;

    /* Save maximal value */
    if (current->match > max_score) {
        max_score = current->match; best_i = i; best_j = j;
    }
    /* Advance pointers for next node */
    if (i > 0) {diag = left; left++;}
    down = current; current++;
}

/* Loop a second time - update only ->del */
down = &mat[i*leny + leny-1]; current = &mat[i*leny];
if (down->match - Y_GO > current->del) current->del = down->match - Y_GO;
for (j = 0; j < leny; j++) {
    if (down->del - Y_GE > current->del) current->del = down->del - Y_GE;
    down = current; current++;
}

/*for (j = leny - 1; j >= 0; j--) {
    for (i = 0; i < lenx; i++)
        printf (" %2d ", mat[i*leny+j].match);
    printf ("\n");
    for (i = 0; i < lenx; i++)
        printf ("%2d %2d ", mat[i*leny+j].ins, mat[i*leny+j].del);
    printf ("\n\n");
}*/

get_wdp_match (x, y, mat, best_i, best_j, 0, ht);
ht->score = max_score;
strcpy (in_y, y); /* y string may have changed cyclically */
free (mat);
free (x);
free (y);
}

/*=====
void get_wdp_match (char x[], char y[], node *mat,
/* Extract the best solution */
{
    char *nice_x, *nice_y, *nice_diff;
    int last_i, j, loc, end, sim_count, id_count, no_n_count;
    int lenx, leny, lower;
    double tmp;
    node *current, *diag, *left, *down, zero;

```

wdp.c

```

zero.match = zero.ins = zero.del = 0;
lenx = strlen (x);
leny = strlen (y);
end = 2*lenx+leny;
if ((nice_x = (char *) malloc (end+2)) == NULL)
    error(1, -1, "Can't allocate memory for nice_x in get_wdp_match\n");
if ((nice_y = (char *) malloc (end+2)) == NULL)
    error(1, -1, "Can't allocate memory for nice_y in get_wdp_match\n");
if ((nice_diff = (char *) malloc (end+2)) == NULL)
    error(1, -1, "Can't allocate memory for nice_diff in get_wdp_match\n");
nice_x[end+1] = nice_y[end+1] = nice_diff[end+1] = '\0';

i = it; j = jt;
last = DIAG;
if (st == 1) last = LEFT;
if (st == 2) last = DOWN;

for (loc = end; last != NONE; loc--) {
    current = &mat[i*leny + j];
    left = &mat[(i-1)*leny + j];
    down = &mat[i*leny + j-1];
    diag = &mat[(i-1)*leny + j-1];
    if (j == 0) {
        down = &mat[i*leny + leny-1];
        diag = &mat[(i-1)*leny + leny-1];
    }
    if (i == 0) diag = left = &zero;

    if (last == DIAG) {
        nice_x[loc] = x[i];
        nice_y[loc] = y[j];
        if (x[i] == y[j]) nice_diff[loc] = '|';
        else nice_diff[loc] = (exact_sc[x[i]-'A'][y[j]-'A'] > 0) ? ' ' : ' ';
        tmp = current->match - exact_sc[x[i]-'A'][y[j]-'A'];
        if (tmp == 0 || i == 0) last = NONE;
        else {
            if (fabs(diag->match - tmp) < EPSILON) last = DIAG;
            else if (fabs(diag->del - tmp) < EPSILON) last = DOWN;
            else if (fabs(diag->ins - tmp) < EPSILON) last = LEFT;
            else printf ("error in back-tracking 4, %d %d\n", i, j);
        }
        if (last == NONE) break;
        i--; j--; if (i < 0) break; if (j < 0) j += leny;
    } else if (last == LEFT) {
        nice_x[loc] = x[i];
        nice_y[loc] = '-';
        nice_diff[loc] = ' ';
        if (current->ins == 0 || i == 0) last = NONE;
        else {
            tmp = current->ins;
            if (fabs(left->match - tmp-X_GO) < EPSILON) last = DIAG;
            else if (fabs(left->ins - tmp-X_GE) < EPSILON) last = LEFT;
            else printf ("error in back-tracking 5, %d %d\n", i, j);
        }
        i--; if (i < 0) break;
    } else if (last == DOWN) {
        nice_x[loc] = '-';
        nice_y[loc] = y[j];
        nice_diff[loc] = ' ';
        if (current->del == 0 || i == 0) last = NONE;
        else {

```

wdp.c

```

tmp = current->del;
if (fabs(down->del - tmp-Y_GE) < EPSILON) last = DOWN;
else if (fabs(down->match - tmp-Y_GO) < EPSILON) last = DIAG;
else printf ("error in back-tracking 6, %d %d\n", i, j);
} j--; if (j < 0) j += leny;
}

ht->xt = it + ht->x0;
ht->x0 = i + ht->x0;
ht->y0 = 0;
ht->yt = leny - 1;
ht->final = 1; ht->type = WRAPAROUND;
ht->len = strlen (nice_y[loc]);
ht->x = subseq (nice_x[loc], 0, ht->len);
ht->y = subseq (nice_y[loc], 0, ht->len);
ht->diff = subseq (nice_diff[loc], 0, ht->len);

id_count = sim_count = no_n_count = 0;
for (i = 0; i < strlen (ht->diff); i++) {
    id_count += (ht->diff[i] == '|');
    if (ht->x[i] != '-' && ht->y[i] != '-')
        sim_count += (exact_sc(ht->x[i]-A')(ht->y[i]-A') > 0);
    if (ht->x[i] != 'N' && ht->y[i] != 'N') no_n_count++;
}

ht->id_percent = 100*(double)(id_count) / (double)(no_n_count);
ht->sim_percent = 100*(double)(sim_count) / (double)(no_n_count);

/* Change Y, put lower/upper case letters in ht->y */
for (i = loc = lower = 0; i < ht->len; i++) {
    if (ht->y[i] != '-') {
        y[loc] = ht->y[i];
        if (lower) ht->y[i] = lowercase (ht->y[i]);
        else ht->y[i] = uppercase (ht->y[i]);
        loc++;
        if (loc == leny) { loc = 0; lower = 1 - lower; }
    }
}

free (nice_x);
free (nice_y);
free (nice_diff);
}

```

```

/*=====**/
hilltop *WillTops (char in_x[], char in_y[], int cutoff, int bound)
/*
 * Dynamic programming routine to match two nucleotide sequences with
 * wraparound of the second sequence, and get ALL distinct peaks, using the
 * HillTops generalization of SW.
 * Scores under bound will not be printed. between two scores above bound
 * only the better one will be printed, unless they are separated by a valley
 * with scores under cutoff.
 * The routine returns the number of alignments found. res will point to a
 * linked list of alignments of that length.
 */

```

```

segment *segp, *seg_tmp;
int i, j, best_i, best_j, best, tmp, segment_flag;
int lenx, leny;
mnode *diag, *left, *down, *current, *col, *prevcol, *tmp_node;

```

wdp.c

```

mnode zero, zero2;
char *x, *y;
hilltop *work, *list, **htpp, *htp;

/* copy input strings to working strings, shorten if necessary */
x = subseq (in_x, 0, MAX_LEN);
y = subseq (in_y, 0, MAX_LEN);
lenx = strlen (x);
leny = strlen (y);
list = work = NULL;

/* allocate memory and initialize matrix vertical edge */
if (!col = (mnode *) malloc ((1+leny * sizeof (struct mnode))) ||
    !prevcol = (mnode *) malloc ((1+leny * sizeof (struct mnode))))
    error(1, -1, "couldn't allocate memory for wrapnodes 2 (%d).\n", leny);

zero.match.s = zero.ins.s = zero.del.s = 0;
zero.match.xy = zero.ins.xy = zero.del.xy = 0;
for (j = 0; j < leny; j++) {
    col[j] = prevcol[j] = zero;
    col[j].match.xy = col[j].ins.xy = col[j].del.xy = (j+1) & 0xffff;
    /* 16 lsbits are for y0, 16 msbits for x0. This is enough assuming
       no ALIGNMENT is longer than 65535 nucleotides */
}

/* main loop */
for (i = 0; i < lenx; i++) {
    /* Advance one column and initialize pointers */
    tmp_node = col; col = prevcol; prevcol = tmp_node;
    left = prevcol; current = col;
    diag = &prevcol[leny-1]; down = &zero;
    segment_flag = 0; segp = NULL;

    /* initialize horizontal edge */
    down->match.xy = down->ins.xy = down->del.xy = (((i+1) & 0xffff) << 16);

    /* Loop over nodes in the column */
    for (j = 0; j < leny; j++) {
        current->match = diag->match;
        if (diag->ins.s > current->match.s) current->match = diag->ins;
        if (diag->del.s > current->match.s) current->match = diag->del;
        current->match.s += exact_sc(x[i]-A')(y[j]-A');
        if (current->match.s <= 0) {
            current->match.s = 0;
            current->match.xy = (((i+1) & 0xffff) << 16) ^ ((j+1) & 0xffff);
        }
        current->ins = left->match;
        current->ins.s -= X_GO;
        if (left->ins.s - X_GE > current->ins.s) {
            current->ins = left->ins; current->ins.s -= X_GE;
        }
        if (current->ins.s <= 0) {
            current->ins.s = 0;
            current->ins.xy = (((i+1) & 0xffff) << 16) ^ ((j+1) & 0xffff);
        }
        current->del = down->match;
        current->del.s -= Y_GO;
    }
}

```

wdp.c

```

if (down->del.s - Y_GO > current->del.s) {
    current->del = down->del; current->del.s -= Y_GO;
}
if (current->del.s <= 0) {
    current->del.s = 0;
    current->del.xy = (((i+1) & 0xffff) << 16) ^ ((j+1) & 0xffff);
}

/* Save transitions of cutoff */
if (!segment_flag) {
    if ((current->match.s) >= cutoff) {
        create_segment(&segp_tmp);
        segp_tmp->next = segp;
        segp->start = segp->best_y = j; segp->end = leny-1;
        segp->score = current->match.s; segp->best_y = j;
        segp->x0 = current->match.xy;
        segment_flag = 1;
    }
} else {
    if (current->match.s > segp->score) {
        segp->score = current->match.s; segp->best_y = j;
        segp->x0 = current->match.xy;
    }
    if ((current->match.s) < cutoff) {
        segp->end = j-1; segment_flag = 0;
    }
}

/* Advance pointers for next node */
diag = left; left++; down = current; current++;

/* loop a second time - update only -del */
down = &col[leny-1]; current = col;
if (down->match.s - Y_GO > current->del.s) {
    current->del = down->match;
    current->del.s -= Y_GO;
}
for (j = 0; j < leny; j++) {
    if (down->del.s - Y_GO > current->del.s) {
        current->del = down->del;
        current->del.s -= Y_GO;
    }
    down = current; current++;
}

/* Deal with segments */
update_all_hilltops (seggp, &work, i, leny);

/* Copy boundaryless hilltops aside */
for (http = &work; http != NULL; ) {
    if ((*http)->boundary != NULL) {
        http = (*http);
        http->type = WRAPAROUND;
        (*http) = ((*http)->next);
        if (http->score > bound)
            w_trim_hilltop (in_x, in_y, http, cutoff);
        if (http->score > bound) { /* Score may have dropped - check again. */
            http->next = list;
            list = http;
        } else destroy_hilltop (&http);
    }
}

```

wdp.c

```

    } else http = &((*http)->next);
}

/* At the end there might be some hilltops which still have boundaries */
for (http = &work; http != NULL; ) {
    http = (*http);
    (*http) = ((*http)->next);
    http->type = WRAPAROUND;
    if (http->score > bound)
        w_trim_hilltop (in_x, in_y, http, cutoff);
    if (http->score > bound) { /* Score may have dropped - check again. */
        destroy_segment_list (&(http->boundary));
        http->next = list;
        list = http;
    } else destroy_hilltop (&http);
}

free (col);
free (prevcol);
free (x);
free (y);
return (list);
}

/*=====
double wdp_score (char in_x[], char in_y[])
/*
 * Wraparound Dynamic programming routine. matches a (long) nucleotide
 * sequence (x) with an unknown number of tandem duplications of another
 * (short) sequence (y). Works in linear memory.
 */
{
    int i, j, best_i, best_j, best, tmp;
    int lenx, leny;
    double max_score = 0.0;
    node *diag, *left, *down, *current, *col, *prevcol, *tmp_node, zero;
    char *x, *y;

    /* copy input strings to working strings, shorten if necessary */
    x = subseq (in_x, 0, MAX_LEN); lenx = strlen (x);
    y = subseq (in_y, 0, MAX_LEN); leny = strlen (y);

    /* allocate memory */
    if (!col = (node *) malloc ((1+leny * sizeof (struct node)))) ||
        !prevcol = (node *) malloc ((1+leny * sizeof (struct node))))
        error(1, -1, "couldn't allocate memory for wrapnodes 3 (%d).\n", leny);

    /* initialize vertical edge */
    zero.match = zero.ins = zero.del = 0;
    for (j = 0; j < leny; j++) col[j] = zero;

    /* main loop */
    for (i = 0; i < lenx; i++) {
        /* Advance one column and initialize pointers */
        tmp_node = col; col = prevcol; prevcol = tmp_node;
        diag = &prevcol[leny-1]; left = prevcol;
        down = &zero; current = col;

        /* Loop over nodes in the column */
    }
}

```

wdp.c

A-243


```

for (j = 0; j < leny; j++) {
    current->match = diag->match;
    if (diag->ins > current->match) current->match = diag->ins;
    if (diag->del > current->match) current->match = diag->del;
    current->match += exact_sc[x[i]-'A'][y[j]-'A'];
    if (current->match <= 0) current->match = 0;

    current->ins = left->match - X_GO;
    if (left->ins - X_GE > current->ins) current->ins = left->ins - X_GE;
    if (current->ins <= 0) current->ins = 0;

    current->del = down->match - Y_GO;
    if (down->del - Y_GE > current->del) current->del = down->del - Y_GE;
    if (current->del <= 0) current->del = 0;

    /* Save maximal value */
    if (current->match > max_score)
        max_score = current->match;
    /* Advance pointers for next node */
    diag = left; left++;
    down = current; current++;
}
/* loop a second time - update only ->del */
down = &col[leny-1]; current = col;
if (down->match - Y_GO > current->del) current->del = down->match - Y_GO;
for (j = 0; j < leny; j++) {
    if (down->del - Y_GE > current->del) current->del = down->del - Y_GE;
    down = current; current++;
}

}
free (col);
free (prevcol);
free (x);
free (y);
return (max_score);
}

/*=====
void w_trim_hilltop (char in_x[], char in_y[], hilltop *ht, int cutoff)
/*
* Trim an alignment and leave just the end if the score (going back) drops
* below 0 at any point. Use a Wraparound matrix.
*/
{
    hilltop *p;
    char *tmpx, *tmpy;
    int i, j;

    tmpx = subseq (in_x, ht->xt, ht->x0);
    if (!tmpx) malloc (strlen (in_y) + 1)))
        error(1, -1, "can't allocate memory in w_trim_hilltop(n)");
    for (i = ht->yt, j = 0; i >= 0; i--)
        tmpy[j++] = inv(in_y[i]);
    for (i = strlen(in_y)-1; i > ht->yt; i--)
        tmpy[j++] = inv(in_y[i]);
    tmpy[j] = '\0';

    w_global_alignment (tmpx, tmpy, &p);
    if (p->xt+1 != strlen (tmpx)) {
        /* printf ("trimming %d (score %2.2f) to %d (score %2.2f). \n", ht->x0,

```

```

        ht->score, ht->xt - p->yt, p->score); */
    ht->x0 = ht->xt - p->yt;
    ht->y0 = ht->yt - p->xt;
    ht->score = p->score;
}

free (tmpx);
free (tmpy);
}

/*=====
void w_global_alignment (char in_x[], char in_y[], hilltop **ht)
/*
* Dynamic programming routine to match two nucleotide sequences, and
* get the end of the best alignment starting at the beginning of them both.
* Returns a Hilltop structure, Runs in linear memory, with wraparound matrix.
*/
{
    segment *seggp, *seg_tmp;
    int i, j, best_i, best_j, max_score, tmp, segment_flag;
    int lenx, leny;
    node *diag, *left, *down, *current, *col, *prevcol, *tmp_node, zero, bad;
    char *x, *y;

    /* copy input strings to working strings, shorten if necessary */
    x = subseq (in_x, 0, MAX_LEN); lenx = strlen (x);
    y = subseq (in_y, 0, MAX_LEN); leny = strlen (y);
    create_hilltop (ht);

    /* allocate memory and initialize matrix vertical edge */
    if (!col = (node *) malloc ((1+leny * sizeof (struct node))) ||
        !prevcol = (node *) malloc ((1+leny * sizeof (struct node))))
        error(1, -1, "couldn't allocate memory for wrapnodes 4 (%d) \n", leny);

    zero.match = zero.ins = zero.del = 0;
    bad.match = bad.ins = bad.del = -1000000;

    for (j = 0; j < leny; j++) col[j] = prevcol[j] = col[j] = bad;

    /* main loop */
    for (i = 0; i < lenx; i++) {
        /* Advance one column and initialize pointers */
        tmp_node = col; col = prevcol; prevcol = tmp_node;
        diag = &prevcol[leny-1]; left = prevcol;
        down = &bad; current = col;
        if (i == 0) diag = &zero;

        /* Loop over nodes in the column */
        for (j = 0; j < leny; j++) {
            current->match = diag->match;
            if (diag->ins > current->match) current->match = diag->ins;
            if (diag->del > current->match) current->match = diag->del;
            current->match += exact_sc[x[i]-'A'][y[j]-'A'];
            if (current->match <= 0) current->match = bad.match;

            current->ins = left->match - X_GO;
            if (left->ins - X_GE > current->ins) current->ins = left->ins - X_GE;
            if (current->ins <= 0) current->ins = bad.ins;

            current->del = down->match - Y_GO;

```



```

if (down->del - Y_GE > current->del) current->del = down->del - Y_GE;
if (current->del <= 0) current->del = bad.del;

/* Save maximal value */
if (current->match > (*ht)->score) {
    (*ht)->score = current->match;
    (*ht)->xt = i;
    (*ht)->yt = j;
}

/* Advance pointers for next node */
diag = left; left++;
down = current; current++;

/* loop a second time - update only ->del */
down = &col[leny-1]; current = col;
if (down->match - Y_GO > current->del) current->del = down->match - Y_GO;
for (j = 0; j < leny; j++) {
    if (down->del - Y_GE > current->del) current->del = down->del - Y_GE;
    down = current; current++;
}

free (col);
free (prevcol);
free (x);
free (y);
}

/*=====
void linear_wrap_alignment (char x[], char y[], hilltop *ht,
                           int by, int bs, int ey, int es)
/*=====*/

/* Dynamic programming routine to match two nucleotide sequences, with
 * wraparound, using linear memory.
 * From the contents of "ht" we figure out whether to compute the whole matrix
 * or to cut above a certain diagonal:
 *
 * -----
 * |*****| the given value of loop_end is the
 * |*****| height of the vertical bar in the drawing.
 * |-----|
 * * bs and es are the beginning and end states. When the routine is run from
 * * the outside they should both be 0 (i.e. match state). Upon recursive calls
 * * either (or both) can also be 1 (insert) or 2 (delete).
 *
 */
{
    int i, j, k, x0, y0, s0, end_x, end_y;
    int lenx, leny, loop_end, h_line, v_line;
    double best_sc;
    lnode *diag, *left, *down, *current, *tmp_node;
    lnode *col, *prevcol, zero, bad;
    hilltop *bottom, *top;

    /* Test on C.elegans clones:
clone.29: CEF10P2 6008 7460 26463 33681 0 3112 96.14
clone.9: CEC08H9 4984 7626 1472 8815 0 1975 92.95
clone.9: CEC08B6 4238 3893 33944 37737 0 1942 97.46
clone.2: CEC0240 2936 2911 29114 31898 0 1089 95.36
clone.43: CEF36H2 759 2178 20965 22923 0 968 89.94
clone.95: CEC2K617 194 2144 9194 11321 0 885 98.32
clone.90: CECW0463 1694 2320 17561 19816 0 784 94.48 */

```

wdp.c

```

/*printf ("In linear alignment x0=%d, xt=%d, y0=%d, yt=%d, bs=%d, es=%d\n",
          ht->x0, ht->xt, ht->y0, ht->yt, bs, es); */

lenx = ht->xt - ht->x0 + 1;
leny = strlen (y);
/* Define h_line */
h_line = (lenx-1) / 2;

/* allocate memory */
if (! (col = (lnode *) malloc (1+leny * sizeof (lnode))))
    error(1, -1, "couldn't allocate memory for nodes in linear alignment.\n");
if (! (prevcol = (lnode *) malloc (1+leny * sizeof (lnode))))
    error(1, -1, "couldn't allocate memory for nodes in linear alignment.\n");

zero.sc[0] = zero.sc[1] = zero.sc[2] = 0.0;
bad.sc[0] = bad.sc[1] = bad.sc[2] = -1000000;
zero.y0[0] = zero.y0[2] = bad.y0[0] = bad.y0[2] = leny-1;
zero.s0[0] = 0; zero.s0[2] = 2; bad.s0[0] = 0; bad.s0[2] = 2;

/* force by and bs as starting point at i == 0 */
if (by < 0) {
    for (j = 0; j < leny; j++)
        col[j] = prevcol[j] = zero;
} else {
    for (j = 0; j < leny; j++) {
        col[j] = bad; prevcol[j] = zero;
        if (j == by) col[j].sc[bs] = 0;
        if (j > by && bs == 2) col[j].sc[2] = col[j-1].sc[2] - Y_GE;
    }
    /* loop a second time - update only ->del */
    if (bs == 2) {
        col[0].sc[2] = col[leny-1].sc[2] - Y_GE;
        for (j = 1; j < by; j++)
            col[j].sc[2] = col[j-1].sc[2] - Y_GE;
    }
}

/* main loop */
best_sc = end_x = end_y = 0;
for (i = 1; i < lenx; i++) {
    /* Advance one column and initialize pointers */
    tmp_node = col; col = prevcol; prevcol = tmp_node;
    diag = &prevcol[leny-1]; left = prevcol;
    down = &zero; current = col;
    down->x0[0] = down->x0[2] = i;

    for (j = 0; j < leny; j++) {
        /* initialize x0, y0 and s0 */
        if (i == h_line) {
            for (k = 0; k < 3; k++) {
                current->x0[k] = i;
                current->y0[k] = j;
                current->s0[k] = k;
            }
        }
        /* update the scores */
        current->sc[0] = diag->sc[0];
        if (diag->sc[1] > current->sc[0]) current->sc[0] = diag->sc[1];
        if (diag->sc[2] > current->sc[0]) current->sc[0] = diag->sc[2];
    }
}

```

wdp.c

```

current->sc[1] = left->sc[0] - X_GO;
if (left->sc[1] - X_GE > current->sc[1])
    current->sc[1] = left->sc[1] - X_GE;

current->sc[2] = down->sc[0] - Y_GO;
if (down->sc[2] - Y_GE > current->sc[2])
    current->sc[2] = down->sc[2] - Y_GE;

/* If necessary - update x0, y0 and s0 */
if (i > h_line) {
    if (current->sc[0] == diag->sc[0]) {
        current->x0[0] = diag->x0[0];
        current->y0[0] = diag->y0[0];
        current->s0[0] = diag->s0[0];
    } else if (current->sc[0] == diag->sc[1]) {
        current->x0[0] = diag->x0[1];
        current->y0[0] = diag->y0[1];
        current->s0[0] = diag->s0[1];
    } else {
        current->x0[0] = diag->x0[2];
        current->y0[0] = diag->y0[2];
        current->s0[0] = diag->s0[2];
    }
}
if (current->sc[1] == left->sc[0] - X_GO) {
    current->x0[1] = left->x0[0];
    current->y0[1] = left->y0[0];
    current->s0[1] = left->s0[0];
} else {
    current->x0[1] = left->x0[1];
    current->y0[1] = left->y0[1];
    current->s0[1] = left->s0[1];
}
if (current->sc[2] == down->sc[0] - X_GE) {
    current->x0[2] = down->x0[0];
    current->y0[2] = down->y0[0];
    current->s0[2] = down->s0[0];
} else {
    current->x0[2] = down->x0[2];
    current->y0[2] = down->y0[2];
    current->s0[2] = down->s0[2];
}
current->sc[0] += exact_sc[x[i+ht->x0]-'A'][y[j]-'A'];
/* advance pointers for next node */
diag = left; left++; down = current; current++;
}

/* loop a second time - update only -del */
down = &col[lenny-1]; current = col;
if (down->sc[0] - Y_GO > current->sc[2]) {
    current->sc[2] = down->sc[0] - Y_GO;
    if (i > h_line) {
        current->x0[2] = down->x0[0];
        current->y0[2] = down->y0[0];
        current->s0[2] = down->s0[0];
    }
}
for (j = 0; j < leny; j++) {
    if (down->sc[2] - Y_GE > current->sc[2]) {

```

```

current->sc[2] = down->sc[2] - Y_GE;
if (i > h_line) {
    current->x0[2] = down->x0[2];
    current->y0[2] = down->y0[2];
    current->s0[2] = down->s0[2];
}
down = current; current++;
}

/* search for best ending point */
if (ey < 0 && i > h_line) {
    for (j = 0; j < leny; j++)
        if (col[j].sc[es] > best_sc) {
            best_sc = col[j].sc[es];
            end_x = j;
            end_y = j;
            x0 = col[j].x0[es] + ht->x0;
            y0 = col[j].y0[es];
            s0 = col[j].s0[es];
        }
}
if (ey >= 0) {
    end_y = ey;
    x0 = col[ey].x0[es] + ht->x0;
    y0 = col[ey].y0[es];
    s0 = col[ey].s0[es];
} else ht->xt = end_x + ht->x0;

create_hilltop (&bottom);
bottom->type = top->type = ht->type;
bottom->x0 = ht->x0; bottom->y0 = ht->y0;
bottom->xt = x0; bottom->yt = y0;
top->x0 = x0; top->y0 = y0;
top->xt = ht->xt; top->yt = ht->yt;
continue_wrap_alignment (x, y, bottom, by, bs, y0, s0);
continue_wrap_alignment (x, y, top, y0, s0, end_y, es);
ht->len = bottom->len + top->len - 1;

if (!ht->x = (char *) malloc (ht->len+1))
    error(1, -1, "couldn't allocate memory for alignment string.\n");
if (!ht->y = (char *) malloc (ht->len+1))
    error(1, -1, "couldn't allocate memory for alignment string.\n");
if (!ht->diff = (char *) malloc (ht->len+1))
    error(1, -1, "couldn't allocate memory for alignment string.\n");

for (i = 0; i < bottom->len; i++) {
    ht->x[i] = bottom->x[i];
    ht->y[i] = bottom->y[i];
    ht->diff[i] = bottom->diff[i];
}
for (i = 0; i < top->len; i++) {
    ht->x[i+bottom->len-1] = top->x[i];
    ht->y[i+bottom->len-1] = top->y[i];
    ht->diff[i+bottom->len-1] = top->diff[i];
}
ht->x[ht->len] = ht->y[ht->len] = ht->diff[ht->len] = '\0';
ht->final = 1;

```

```

destroy_hilltop (&bottom);
destroy_hilltop (&stop);
free (col);
free (prevcol);
}

/*=====
void continue_wrap_alignment (char in_x[], char in_y[], hilltop *ht,
                             int by, int bs, int ey, int es)
/*
* Wraparound Dynamic programming routine. matches a (long) nucleotide
* sequence (x) with an unknown number of tandem duplications of another
* (short) sequence (y), then finds alignment. Works in square memory.
* It is called by "linear_wrap_alignment", and calls it back (if necessary).
*/
{
    int i, j, best_i = 0, best_j = 0, tmp;
    int lenx, leny, max_score = 0;
    node *diag, *left, *down, *current, *mat, zero, bad, good;
    char *x, *y;

    /* copy input strings to working strings, shorten if necessary */
    lenx = ht->xt - ht->x0 + 1;
    leny = strlen (in_y);

    /* allocate memory */
    if (lenx*leny > MAX_FULL*MAX_FULL) {
        linear_wrap_alignment (in_x, in_y, ht, by, bs, ey, es);
        return;
    }

    if (!mat = (node *) malloc ((1+lenx*leny * sizeof (node))))
        error (1, -1, "Can't allocate memory for nodes.\n");
    x = subseq (in_x, ht->x0, ht->xt);
    y = subseq (in_y, 0, leny);

    zero.match = zero.ins = zero.del = 0;
    bad.match = bad.ins = bad.del = -1000000;
    good.match = good.ins = good.del = 100;

    /* force by and bs as starting point at i == 0 */
    if (by < 0) {
        for (j = 0; j < leny; j++)
            mat[j] = good;
    } else {
        for (j = 0; j < leny; j++) {
            mat[j] = bad;
            if (j == by && bs == 0) mat[j].match = 1000;
            if (j == by && bs == 1) mat[j].ins = 1000;
            if (j == by && bs == 2) mat[j].del = 1000;
            if (j > by && bs == 2) mat[j].del = mat[j-1].del - Y_GE;
        }
    }

    /* loop a second time - update only ->del */
    if (bs == 2) {
        mat[0].del = mat[leny-1].del - Y_GE;
        for (j = 1; j < by; j++)
            mat[j].del = mat[j-1].del - Y_GE;
    }
}

```

```

)

/* main loop */
for (i = 1; i < lenx; i++) {
    /* Advance one column and initialize pointers */
    diag = &mat[(i-1)*leny + leny-1]; left = &mat[(i-1)*leny];
    down = &zero; current = &mat[i*leny];

    /* Loop over nodes in the column */
    for (j = 0; j < leny; j++) {
        current->match = diag->match;
        if (diag->ins > current->match) current->match = diag->ins;
        if (diag->del > current->match) current->match = diag->del;
        current->match += exact_sc[x[i]-'A'][y[j]-'A'];

        current->ins = left->match - X_GO;
        if (left->ins - X_GE > current->ins) current->ins = left->ins - X_GE;

        current->del = down->match - Y_GO;
        if (down->del - Y_GE > current->del) current->del = down->del - Y_GE;

        /* Advance pointers for next node */
        diag = left; left++; down = current; current++;
    }

    /* loop a second time - update only ->del */
    down = &mat[i*leny + leny-1]; current = &mat[i*leny];
    if (down->match - Y_GO > current->del) current->del = down->match - Y_GO;
    for (j = 0; j < leny; j++) {
        if (down->del - Y_GE > current->del) current->del = down->del - Y_GE;
        down = current; current++;
    }

    get_wdp_match (x, y, mat, lenx-1, ey, es, ht);
    /* y may have changed cyclically, but we ignore it */
    free (mat);
    free (x);
    free (y);
}

```

Sun Aug 9 10:39:56 1998

Listing for Adam Sartiell

```

#define MAX_FULL 1200
#define MAX_LEN 250000
#define MAX_SEG 2500
#define TANDEM_GAP 20
#define EPSILON 0.00001

#define BEST -1
#define CORNER -2

#include "hilltops.h"

void init_matrix (int match, int mismatch, int tA, int tB);

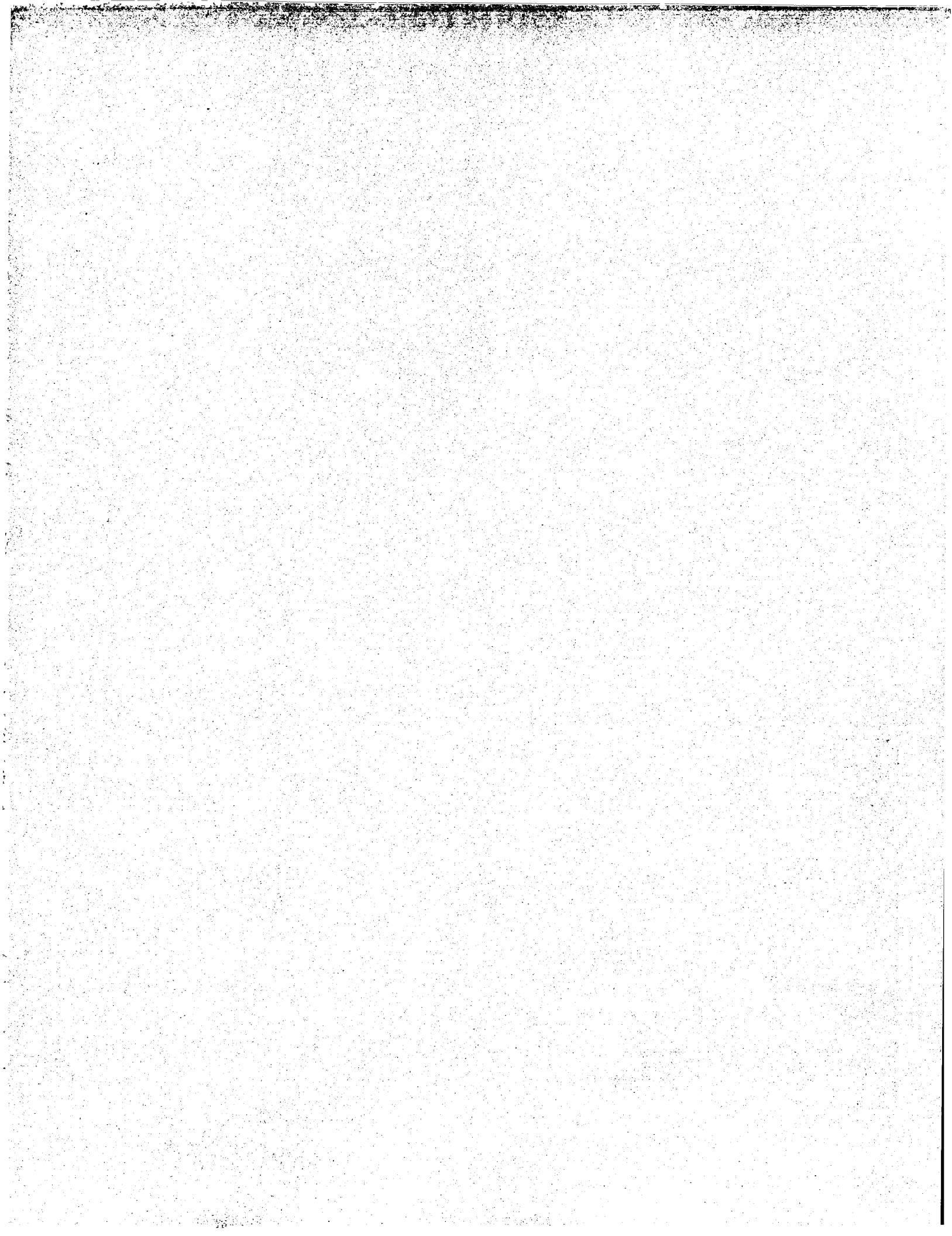
hilltop *HillTops (char in_x[], char in_y[], int mode, int cutoff, int bound);
double smith_waterman (char in_x[], char in_y[]);
void global_alignment (char in_x[], char in_y[], int loop_end, hilltop **ht);
void get_alignment (char in_x[], char in_y[], hilltop *ht, int where);
void linear_alignment (char in_x[], char in_y[], hilltop *ht, int bs, int es);
hilltop *get_six18_alignment (char in_x[], char in_y[], int mode);
hilltop *band_sw (char x[], char y[], int x0, int y0, int width);

void print_alignment (hilltop data);

/* Structure definitions for hilltops */
typedef struct mem {
    int s;
    unsigned int xy;
} mem;

typedef struct mmode {
    mem match, ins, del;
} mmode;

```



```

#define HILLTOPS_INC
#define HILLTOPS_INC

#define STRAIGHT -1
#define INVERTED -2
#define TANDEM -3
#define WRAPAROUND -4
#define UNDEFINED -5

#define NAME_LEN 20

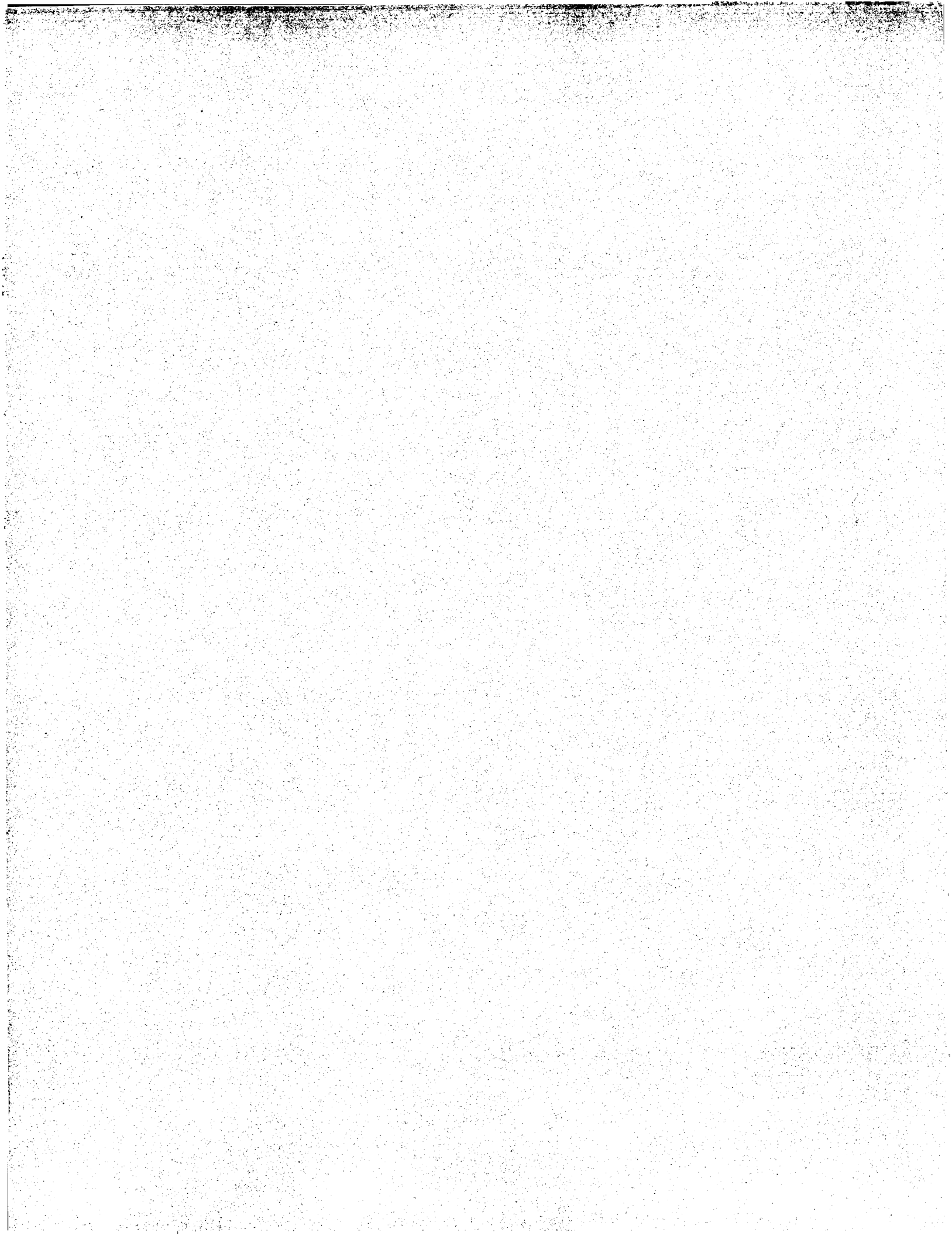
typedef struct segment {
    struct segment *next;
    int start, end;
    double score;
    int best_y;
    int x0, y0;
} segment;

typedef struct hilltop {
    struct hilltop *next;
    int type;
    char name[NAME_LEN]; /* name of repeat */
    int final; /* flag: 1 = final alignment, 0 = under work */
    char *x, *y; /* output sequences */
    char *diff; /* difference between output sequences */
    double id_percent; /* percentage of identity in the alignment */
    double sim_percent; /* percentage of similarity in the alignment */
    int len; /* length of output sequences (identical) */
    int x0, y0; /* position of beginning of alignment in input */
    int xt, yt; /* position of end of alignment in input */
    double score; /* score of match */
    int area; /* Hilltop area */
    segment *boundary; /* linked list of segments */
} hilltop;

void create_segment (segment **obj);
void destroy_segment (segment **obj);
void destroy_segment_list (segment **obj);
void copy_segment (segment from, segment *obj);
void create_hilltop (hilltop **obj);
void destroy_hilltop (hilltop **obj);
void copy_hilltop (hilltop from, hilltop *obj);
void update_all_hilltops (segment *seg_list, hilltop **ht_list, int x, int w);
void update_hilltop (hilltop *htp, segment seg, int x);
void merge_hilltops (hilltop *htp1, hilltop *htp2);
void trim_hilltop (char in_x[], char in_y[], hilltop *ht, int cutoff);
char *subseq (char *seq, int first, int last);
char inv (char c);

#endif

```

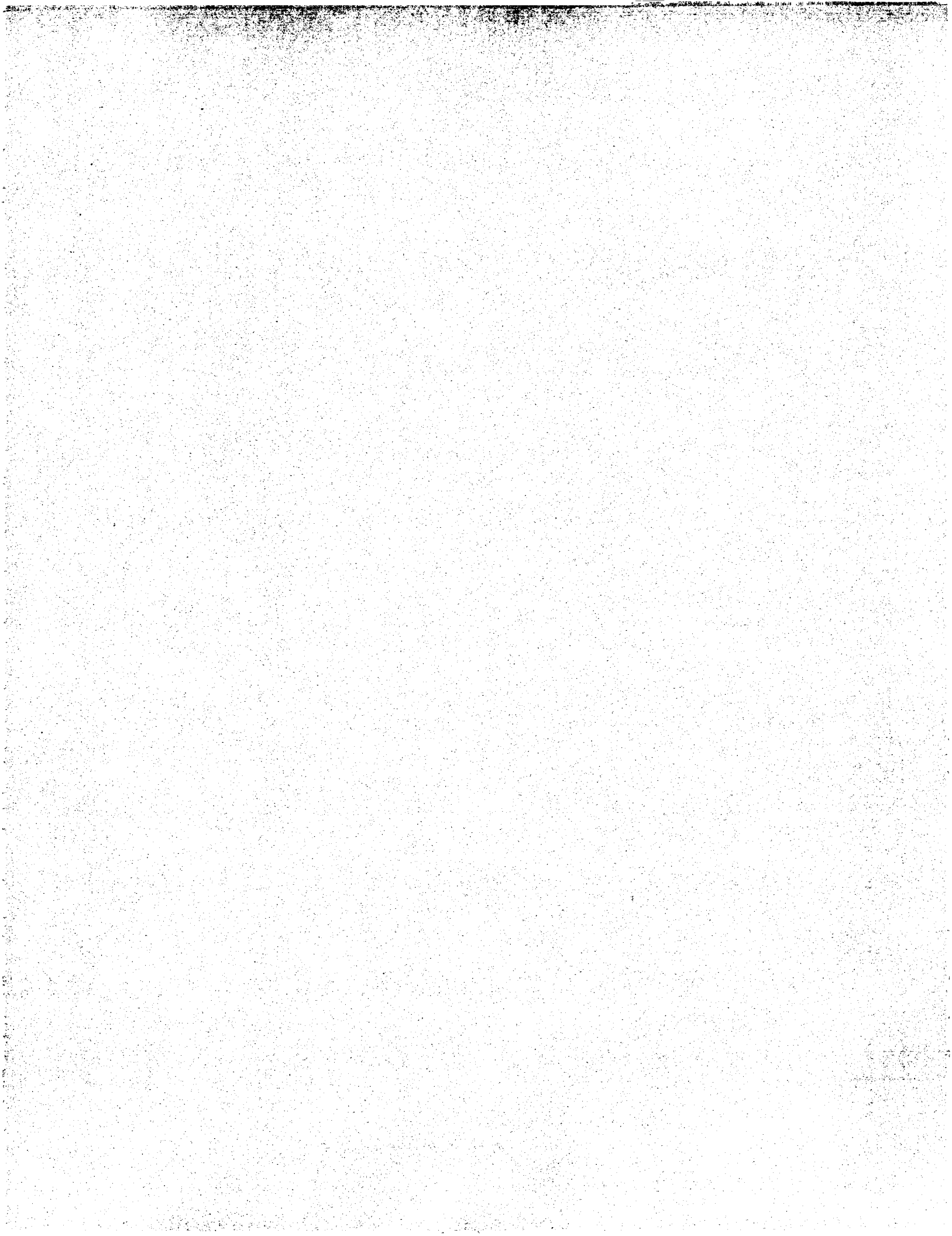


Sun Aug 9 10:39:56 1998

Listing for Adam Santel

```
hilltop *WhillTops (char in_x[], char in_y[], int cutoff, int bound);  
void get_wrap_alignment (char in_x[], char in_y[], hilltop *ht);  
double wdp_score (char in_x[], char in_y[]);
```

wdp.h



```
CC = cc
CFLAGS = -c -O -I$(PRODR00T)/include
CL = cc

LFLAGS = -L$(PRODR00T)/lib$(ARCH)obj -lm
LPRM = -lprn

SRCS = cln.c dp.c wdp.c hilltops.c
OBJS = cln_$(ARCH).o dp_$(ARCH).o wdp_$(ARCH).o hilltops_$(ARCH).o
all: cln_$(ARCH)

cln_$(ARCH): $(OBJS) Makefile
$(CL) $(CFLAGS) $(LFLAGS) $(LPRM) -o cln_$(ARCH)

cln.c hilltops.h Makefile
$(CC) $(CFLAGS) cln.c -o cln_$(ARCH).o
dp.c dp.h Makefile
$(CC) $(CFLAGS) dp.c -o dp_$(ARCH).o
wdp.c wdp.h dp.h hilltops.h Makefile
$(CC) $(CFLAGS) wdp.c -o wdp_$(ARCH).o
hilltops.c hilltops.h Makefile
$(CC) $(CFLAGS) hilltops.c -o hilltops_$(ARCH).o

depend : $(SRCS)
makedepend $(CFLAGS) $(SRCS)

clean: rm *$(ARCH).o cln_$(ARCH) core

install: cln_$(ARCH)
cp cln_$(ARCH) $(PRODR00T)/bin/$(ARCH)/cln

# DO NOT DELETE THIS LINE -- make depend depends on it.

cln.o: /usr/include/stdio.h /usr/include/sys/feature_tests.h
cln.o: /usr/include/sys/valist.h /usr/include/malloc.h /usr/include/math.h
cln.o: /home/prod/include/prn.h hilltops.h wdp.h
dp.o: /usr/include/stdio.h /usr/include/sys/feature_tests.h
dp.o: /usr/include/sys/valist.h /usr/include/strings.h
dp.o: /usr/include/sys/types.h /usr/include/sys/isa_defs.h
dp.o: /usr/include/sys/machtypes.h /usr/include/sys/int_types.h
dp.o: /usr/include/sys/select.h /usr/include/sys/time.h
dp.o: /usr/include/sys/time.h /usr/include/string.h /usr/include/malloc.h
dp.o: /usr/include/math.h dp.h hilltops.h
wdp.o: /usr/include/stdio.h /usr/include/sys/feature_tests.h
wdp.o: /usr/include/sys/valist.h /usr/include/strings.h
wdp.o: /usr/include/sys/types.h /usr/include/sys/isa_defs.h
wdp.o: /usr/include/sys/machtypes.h /usr/include/sys/int_types.h
wdp.o: /usr/include/sys/select.h /usr/include/sys/time.h
wdp.o: /usr/include/string.h /usr/include/malloc.h
hilltops.o: /usr/include/stdio.h /usr/include/sys/feature_tests.h
hilltops.o: /usr/include/sys/valist.h /usr/include/strings.h
hilltops.o: /usr/include/sys/types.h /usr/include/sys/isa_defs.h
```



```

)
/*=====
void init_bins (bin_t **bins, int n)
/* Initialize bin pointers */
{
    if (!(*bins = (bin_t **)calloc (n, sizeof (bin_t *))))
        error (1, -1, "Can't allocate bin pointers.\n");
}

/*=====
void clear_bins (bin_t **bins, int n)
/* free all memory from bins */
{
    int i;
    bin_t *b;
    for (i = 0; i < n; i++)
        bins[i] = NULL;
}

/*=====
int throw_to_bin (mword bin_no, unsigned short seq_no, unsigned short loc,
                 bin_t **bins, unsigned short inv, int max, int newpass)
{
    static int used, first = 1, oldpass = 0;
    static bin_t *array;
    bin_t *b;
    char str[10];
    static int num_of_entry = 0, seq_num = 0;

    if (first) {
        first = used = 0;
        if (!(array = (bin_t *) malloc (max * (sizeof (bin_t)))))
            error (1, -1, "Can't allocate memory for bins.\n");
    }

    if (newpass != oldpass) {
        used = 0;
        oldpass = newpass;
    }

    b = &array[used++];
    if (used == max) return 0;
    b->next = bins[bin_no];
    bins[bin_no] = b;
    b->seq_no = seq_no;
    b->loc = loc;
    b->inv = inv;
    return 1;
}

/*=====
void get_sequence (char *str, int seq_no, int inv)
{
    int len;
    char *st2;
    mword word;

    len = get_comp_length (seq_no);
    get_est (seq_no, str);
    if (inv) {
        st2 = subseq (str, len, 0);

```

cluster.c

```

strcpy (str, st2);
free(st2);
}

/*=====
char *subseq (char *seq, int first, int last)
/* Cut a sub-sequence from a given input sequence. If first > last, flip it
 * and translate C->G, A->T, (and subsets as well).
*/
{
    char *res;
    int i, len, newlen, flip = 0;

    len = strlen (seq);

    if (first < 0) first = 0;
    if (first >= len) first = len-1;
    if (last < 0) last = 0;
    if (last >= len) last = len-1;

    newlen = last - first;
    if (newlen < 0) {
        flip = 1;
        newlen = -newlen;
    }
    newlen++;
    if (!(res = (char *) malloc (newlen+1)))
        error (1, -1, "subseq: can't allocate string\n");

    res[newlen] = '\0';
    if (flip)
        for (i = 0; i < newlen; i++)
            res[i] = inv(seq[first-i]);
    else
        for (i = 0; i < newlen; i++)
            res[i] = seq[first+i];

    return (res);
}

/*=====
char inv (char c)
/* return base pair
*/
{
    char alphabet[] = "ACMGRSVWYHKDBN";
    /* K = G or T M = A or C R = G or A S = G or C W = A or T Y = T or C
    * B = G or T or C D = G or A or T H = A or C or T V = G or C or A
    * N = A or C or G or T */
    if (c == 'A') return 'T';
    if (c == 'C') return 'G';
    if (c == 'G') return 'C';
    if (c == 'T') return 'A';
    if (c == '-') return '-';
    if (c == 'N') return 'N';
    if (c == 'R') return 'Y';
    if (c == 'Y') return 'R';

```

cluster.c

A-257

```

    free (bins);
}
/*=====
void third_pass(bin_t *bin)
{
    bin_t *b, *c;
    int len = 0, inv, loc1, loc2, ok1, ok2, bin_no;
    char str[20];
    int r = 0, tot = 0, suc = 0;
    int x0, y0, xt, yt, mis;
    static char *seq1, *seq2, first = 1;

    /* Do the malloc only once */
    if (first == 1) {
        seq1 = (char *) malloc ( MAX_EST_SIZE );
        seq2 = (char *) malloc ( MAX_EST_SIZE );
        first = 0;
    }

    /* Go over all sequence pairs in the bin */
    for (b = bin; b != NULL; b = b->next) {
        seq1[0] = '\0'; /* Mark we still didn't read the sequence */
        loc1 = b->loc;
        if (b->inv) loc1 += tuple_size;
        for (c = b->next; c != NULL; c = c->next) {
            counter_pairs++;
            loc2 = c->loc;
            if (c->inv) loc2 += tuple_size;
            /* Check the pair isn't already united */
            /* If (UF_find(union_find,b->seq_no)!=UF_find(union_find,c->seq_no)) {
                if (fast_uf_fathers[b->seq_no] != fast_uf_fathers[c->seq_no]) {
                    inv = b->inv ^ c->inv;
                    counter_check++;
                    loc2 = inv ? get_comp_length (c->seq_no, 0);
                    /* Check that we haven't come across this pair before */
                    if (find_in_CQ (b->seq_no, c->seq_no, loc1-loc2, inv, cq) == -1) {
                        /* Get the second sequence, and the first (if we haven't already) */
                        get_sequence (seq2, c->seq_no, inv);
                        if (seq1[0] == '\0') get_sequence (seq1, b->seq_no, 0);
                        counter_sw++;
                        /* Test the pair. If good - print and unite, if bad - remember */
                        if (fast_half_overlap (seq1, seq2, get_type (b->seq_no),
                            &x0, &xt, &y0, &yt, &mis) == 1) {
                            counter_merge++;
                            /* UF_union (union_find, b->seq_no, c->seq_no); */
                            fast_uf_union (b->seq_no, c->seq_no);
                            fprintf (pairs_file, "%s %s %c %d %d %d %d\n",
                                get_name (b->seq_no), get_name (c->seq_no),
                                inv ? '-' : '+', x0, xt, y0, yt, mis);
                            fflush (pairs_file);
                        } else {
                            counter_cq++;
                            cq_pos = insert_to_CQ (b->seq_no, c->seq_no, loc1-loc2, inv, cq);
                            cq_inserts++;
                        }
                    }
                }
            }
        }
    }
}

```

```

}
/*=====
void edge_third_pass(bin_t *bin)
{
    bin_t *b, *c;
    int len = 0, inv, loc2, loc1, ok1, ok2, bin_no;
    char str[20];
    int r = 0, tot = 0, suc = 0;

    for (b = bin; b != NULL; b = b->next) {
        loc1 = b->loc;
        if (b->inv)
            loc1 += tuple_size;
        ok1 = (loc1 < EDGE_LEN ||
            loc1 > get_comp_length (b->seq_no)-EDGE_LEN);
        for (c = b->next; c != NULL; c = c->next) {
            loc2 = c->loc;
            if (c->inv)
                loc2 += tuple_size;
            /* If (UF_find(union_find,b->seq_no)!=UF_find(union_find,c->seq_no)) {
                if (fast_uf_fathers[b->seq_no] != fast_uf_fathers[c->seq_no]) {
                    counter_check++;
                    inv = b->inv ^ c->inv;
                    ok2 = (loc2 < EDGE_LEN ||
                        loc2 > get_comp_length (c->seq_no) - EDGE_LEN);
                    if (ok1 || ok2) {
                        loc2 = inv ? get_comp_length (c->seq_no) - loc2 + 1 : loc2;
                        counter_merge++;
                        /* UF_union (union_find, b->seq_no, c->seq_no); */
                        fast_uf_union (b->seq_no, c->seq_no);
                        fprintf (pairs_file, "%s %s %d %d %c\n",
                            get_name (b->seq_no), get_name (c->seq_no),
                            get_type (b->seq_no), get_type (c->seq_no),
                            inv ? '-' : '+');
                        fflush (pairs_file);
                    }
                }
            }
        }
    }
}
/*=====
void fast_third_pass(bin_t *bin)
{
    bin_t *b, *c;
    int inv;

    b = bin;
    for (c = b->next; c != NULL; c = c->next) {
        /* If (UF_find(union_find,b->seq_no)!=UF_find(union_find,c->seq_no)) {
            if (fast_uf_fathers[b->seq_no] != fast_uf_fathers[c->seq_no]) {
                inv = b->inv ^ c->inv;
                counter_merge++;
                /* UF_union (union_find, b->seq_no, c->seq_no); */
                fast_uf_union (b->seq_no, c->seq_no);
                fprintf (pairs_file, "%s %s %c\n",
                    get_name (b->seq_no), get_name (c->seq_no), inv ? '-' : '+');
            }
        }
    }
}

```

```

    bin_t      *b, **bins, tmp_element, *array, *b_chk;
    HS_int_struct h;
    mword      key, inv_key;
    int         bin_no, bin_no2, tmp, dist, len, i, j, flag, mem = 0;
    int         new_loc;
    unsigned short short_loc, inv;
    el_int      *hash_element;
    char         str[20];
    int         compare;

    /* lookup bin size */
    for (b = bin, len = 0; b != NULL; b = b->next) len++;
    if (len < 2) return;

    *bin_no = get_comp_key (bin->seq_no, bin->loc, tuple_size);
    bin_no2 = invert_key (bin_no, tuple_size);
    word_to_chars (str, (bin->inv) ? bin_no2 : bin_no, tuple_size);
    str[tuple_size] = '\0';
    printf("%6d (%6d) %s", (bin_no < bin_no2) ? bin_no : bin_no2, len, str);
    fflush (stdout); /**/

    /* Allocate memory */
    h = init_int_hash (len*max_dist);
    init_bins (&bins, len*max_dist);
    if (!array = (bin_t *) malloc (len*max_dist*(sizeof (bin_t))))
        error (1, -1, "Can't allocate memory for %d-level bins.\n", depth);

    /* Go over the entries in the bin */
    for (b = bin, j = 0; b != NULL; b = b->next) {
        /* Try all allowed distances */
        for (dist = 0; dist < max_dist; dist++) {
            if (b->inv) {
                /* Inverted */
                new_loc = b->loc - tuple_size - dist;
                if (new_loc < 0) continue;
                short_loc = (unsigned short) (new_loc & 0xffff);
                key = get_comp_key (b->seq_no, short_loc, tuple_size);
                key = invert_key (key, tuple_size);
            } else {
                /* Straight */
                new_loc = b->loc + tuple_size + dist;
                short_loc = (unsigned short) (new_loc & 0xffff);
                key = get_comp_key (b->seq_no, short_loc, tuple_size);
                if (key == -1) continue;
            }
            inv_key = get_invariant_key (b->seq_no, short_loc, tuple_size, &inv);
            /* ===== */
            /* THESE ARE THE ORIGINAL LINES -CHANGED ON 7/4/98 BEFORE GENBANK106 */
            if (((b->inv ^ inv) && (inverted_sizes[inv_key] >= 0)) ||
                (((b->inv ^ inv) && (non_inverted_sizes[inv_key] >= 0))) {

                /* THESE SHOULD BE USED FOR ERAN'S CHANGE */

                /* if (((b->inv ^ inv) && (inverted_sizes[inv_key] >= -1)) ||
                    (((b->inv ^ inv) && (non_inverted_sizes[inv_key] >= -1))) { */

```

cluster.c

```

    /* ===== */
    /* Throw balls to the next-level bins */
    tmp = HS_int_find_or_insert (h, key);
    array[mem].next = bins[tmp];
    bins[tmp] = &array[mem];
    array[mem].seq_no = b->seq_no;
    array[mem].loc = short_loc;
    array[mem].inv = b->inv;
    if (mem++ >= len*max_dist) error (1, -1, "Exceeded memory limits\n");
}

/* Go over the bins and do the next level (or next pass) */
if (depth == max_depth)
    for (flag = 0, hash_element = NULL; !flag; ) {
        hash_element = hash_next_element(h, hash_element, &i);
        if (hash_element == NULL) flag = 1;
    } else {
        /* here we gain the 2 factor,
           * we continue only if current bin is bigger than the
           * first bin that created it.
           */
        b_chk = bins[hash_element->data];
        compare = get_comp_key (b_chk->seq_no, b_chk->loc, tuple_size);
        if (b_chk->inv) {
            compare = invert_key (compare, tuple_size);
        }
        if (compare >= first_key_straight) {
            if (sw_flag)
                third_pass (bins[hash_element->data]);
            else if (edge_flag)
                edge_third_pass (bins[hash_element->data]);
            else
                fast_third_pass (bins[hash_element->data]);
        }
    }
    else
        for (flag = 0, hash_element = NULL; !flag; ) {
            hash_element = hash_next_element(h, hash_element, &i);
            if (hash_element == NULL) flag = 1;
        } else {
            second_pass (bins[hash_element->data], depth+1);
        }
}

/*for (i = 0; i < tuple_size + 17; i++)
    printf ("%b", */

/* Free all memory allocations */
HS_int_destroy (h);
free (array);

```

cluster.c

```

printf ("Using %d bins every pass, estimating %d passes\n",
        N_bins, N_passes);

N_bins = 2 * N_bins;
init_bins (&bins, N_bins);
printf ("bins are initialized\n");

/* Start the N_passes passes */
min = first;
loop_end = ((last < 0) ? total_bins : last);
i = 0;
for (key1 = 0, tot = 0; (tot + i < N_entry && key1 <= loop_end); key1++) {
    tot += i;
    i = 0;
    if (inverted_sizes[key1] >= 0)
        i += inverted_sizes[key1];
    if (non_inverted_sizes[key1] >= 0)
        i += non_inverted_sizes[key1];
}
printf (" tot is %lu N_entry id %d\n", tot, N_entry);
if (key1 < 2)
    key1 = 2;
max = MIN(key1 - 2, loop_end);
min = MIN(max, min + N_bins - 1);
for (pass = 0; min < loop_end; pass++) {
    if (max == min)
        error (1, -1, "there is not enough memory for bin %d\n", min);
    printf ("\npass %d, min = %d, max = %d\n", pass, min, max);
    printf ("Going over data...");
    fflush (stdout);
    clear_bins (bins, N_bins);
    out_of_memory = 0;

    /* Go over the sequences */
    for (seq_no = 0; seq_no < N_seq; seq_no++) {
        /* Go over locations in the sequence */
        len = get_comp_length (seq_no);
        len = MIN(len, MAX_SEQ_SIZE);
        for (loc = 0; loc < len - tuple_size + 1; loc++) {
            key1 = get_invariant_key (seq_no, loc, tuple_size, &inv);
            if (key1 >= min && key1 <= max) {
                if ((inv) && (inverted_sizes[key1] >= 0)) ||
                    ((inv) && (non_inverted_sizes[key1] >= 0)) {
                    if (!throw_to_bin (key1 - min, seq_no, loc, bins, inv,
                                        N_entry, pass & 1)) {
                        out_of_memory = 1;
                        break;
                    }
                }
            }
        }
        if (out_of_memory) break;
    }
    if (!out_of_memory) {
        printf ("Done.\n");
        call_second_pass (bins, min, max);
        min = max + 1;
        i = 0;
        for (key = min, tot = 0; tot + i < N_entry && key <= loop_end; key++) {
            tot += i;

```

```

i = 0;
if (inverted_sizes[key] >= 0)
    i += inverted_sizes[key];
if (non_inverted_sizes[key] >= 0)
    i += non_inverted_sizes[key];
}
if (key < min + 2)
    key = min + 2;
max = MIN(key - 2, loop_end);
min = MIN(max, min + N_bins - 1);
} else {
    max = (max + min) / 2;
    printf ("again...");
    fflush (stdout);
}
}

/* next parameters are for the factor 2 speed_up */
int first_key_straight=0;
int first_key_inverse;

/*=====
void call_second_pass (bin_t *bins, int min, int max)
{
    int i, len;
    bin_t *b;
    int tmp;

    for (i = min; i <= max; i++) {
        printf ("bin %d pair %d chk %d sw %d mrg %d cq %d\n", i,
                counter_pairs, counter_check, counter_sw, counter_merge, counter_cq);
        fflush (stdout);
        b = bins[i - min];

        if (b != NULL) {
            tmp = get_comp_key (b->seq_no, b->loc, tuple_size);
            if (b->inv) {
                tmp = invert_key (tmp, tuple_size);
            }
            first_key_straight = tmp;

            second_pass (bins[i - min], 2);
            for (b = bins[i - min]; b != NULL; b = b->next)
                b->inv ^= 1;

            first_key_straight = invert_key (first_key_straight, tuple_size);
            second_pass (bins[i - min], 2);
        }
    }

    /*=====
void second_pass (bin_t *bin, int depth)
{

```

A-254

```

"-local ~
EOLIST);

! sticking local overlaps", &local_flag,

if (local_flag & only_full_overlap)
    error (1, -1, "Choose either -local or -only_full_overlap, not both.\n");
if (edge_flag & sw_flag)
    error (1, -1, "Choose either -edge or -sw, not both.\n");
if (!tuple_size & 1) error (1, -1, "Please enter an odd k only!\n");
init_alphabet("ACGT");

sprintf (name, "%s/%s", dir, fname);
if (!seqfile = fopen (name, "r")) == NULL)
    error (1, -1, "Can't open sequence file %s\n", name);
sprintf (name, "%s/%s.hash", dir, fname);
if (!fp_hash = fopen (name, "w")) == NULL)
    error (1, -1, "Can't open sequence file %s\n", name);
sprintf (pairname, "%s/%s.pairs", dir, fname);
if (!pairs_file = fopen (pairname, "w")) == NULL)
    error (1, -1, "Can't open pairs file %s\n", pairname);
}

/*=====*/
void check_bin_sizes()
{
    int len, total_bins, bad_p, max;
    unsigned int seq_no, i;
    unsigned short inv, loc;
    mword key, key2;
    char str[20];

    total_bins = (1 << (2*tuple_size-1));
    non_inverted_sizes = (int *) malloc (total_bins * sizeof (int));
    inverted_sizes = (int *) malloc (total_bins * sizeof (int));
    for (i = 0; i < total_bins; i++) {
        non_inverted_sizes[i] = 0;
        inverted_sizes[i] = 0;
    }
    printf("\n");
    for (seq_no = 0; seq_no < N_seq; seq_no++) {
        /* Go over locations in the sequence */
        if (!seq_no & 1023)
            printf("%d\r", seq_no);
        fflush (stdout);
        len = get_comp_length (seq_no);
        len = MIN (len, MAX_SEQ_SIZE);
        for (loc = 0; loc < len - tuple_size + 1; loc++) {
            key = get_invariant_key (seq_no, loc, tuple_size, &inv);
            if (inv) inverted_sizes[key]++;
            else non_inverted_sizes[key]++;
        }
    }
    /* code to produce bad_bin table */
    for (key = i = 0; key < (1 << (2*tuple_size-1)); key++) {
        if (inverted_sizes[key] > max_bin_size)
            printf ("%d %05x i\n", i++, key);
        if (non_inverted_sizes[key] > max_bin_size)
            printf ("%d %05x 0\n", i++, key);
    }
    /* Signify all the known bad bins (low complexity tuples) as overfull */

```

cluster.c

```

for (bad_p = 0; bad_p < BAD_BIN_NO; bad_p++) {
    if (bad_bins[bad_p][1]) inverted_sizes[bad_bins[bad_p][0]] = -1;
    else
        non_inverted_sizes[bad_bins[bad_p][0]] = -1;
}
for (key = max = 0; key < (1 << (2*tuple_size-1)); key++) {
    if (inverted_sizes[key] > max) max = inverted_sizes[key];
    if (non_inverted_sizes[key] > max) max = non_inverted_sizes[key];
}
printf ("Biggest bin is %d locations.\n", max);
fflush (stdout);
}

/*=====*/
void check_data_length (FILE *seqfile)
{
    char st[MAX_LINE_SIZE];
    double max;

    rewind (seqfile);
    N_seq = N_base = 0;

    do {
        if (fgetc (st, MAX_LINE_SIZE, seqfile) == NULL)
            break;
        if (st[0] == '>') N_seq++;
        else N_base += (strlen (st) - 1);
    } while (FOREVER);

    max = N_base*2/(1 << 2*tuple_size);
    max += sig * sqrt (max);
    max_bin_size = (int) max + 30;
    printf ("Max_bin_size is %d.\n", max_bin_size);
}

/*=====*/
void first_pass ()
{
    int i, total_bins, N_bins, N_passes, N_entry, mask;
    int max, pass, len, type, flag, out_of_memory, loop_end;
    unsigned int seq_no;
    unsigned short loc, inv;
    mword key1, key2, min, key, tot;
    double bytes_per_bin;
    char st[MAX_TUPLE], name[MAX_NAME_LEN];
    mword seq[MAX_LINE_SIZE];
    bin_t *bins;
    unsigned int *bins_pointers;
    FILE *binfile;
    char str[20];

    mask = (1 << 2*tuple_size) - 1;
    printf ("There are %d bytes in a bin-entry.\n", sizeof (bin_t));

    total_bins = (1 << (2*tuple_size-1));
    bytes_per_bin = 8 + ((double)N_base* sizeof (bin_t))/total_bins;
    N_entry = mem_size*1024*1024 / sizeof (bin_t);
    N_bins = MIN(mem_size*1024*1024 / bytes_per_bin, total_bins);
    N_passes = (int) ((double)total_bins/N_bins + 0.5);

```

cluster.c


```

#include <math.h>
#include "prm.h"
#include "cluster.h"
#include "hash_int.h"
#include "union_find.h"
#include "comp_pack.h"
#include "lib.h"
#include "bad_bins.h"
#include "cyclic_queue.h"

#define MAX_LINE_SIZE 30000
#define MAX_TUPLE 20
#define MAX_NAME_LEN 20
#define MAX_SEQ_SIZE 60000
#define MAX_EST_SIZE 150000

#define EDGE_LEN 20
#define FOREVER 1

/* global arguments */
UP_struct *union_find;
CQ_struct *cq;
int bound, cutoff, gapext, gapop, match, mismatch, rm_flag, memory_used;
int total_len = 0, counter_sw = 0, counter_merge = 0, counter_check = 0;
int cq_inserts = 0, counter_cq = 0;
long counter_pairs = 0;
int only_full_overlap, local_flag, cq_pos;
FILE *pairs_file, *fp_hash;
int *non_inverted_sizes, *inverted_sizes;
int *fast_uf_fathers, *fast_uf_brothers, *fast_uf_size;

/* These are the famous n, m, k, M, g, and t */
int N_seq, N_base, tuple_size, mem_size, max_size, max_dist, max_depth;
int first, last, max_bin_size, sw_flag, edge_flag, sig;

/* function prototypes */
void prologue(int argc, char *argv[], FILE **seqfile);
void check_data_length(FILE *seqfile);
void first_pass();
void init_bins(bin_t **bins, int n);
void clear_bins(bin_t **bins, int n);
void throw_to_bin(mword bin_no, unsigned int seq_no, unsigned short loc);
void call_second_pass(bin_t **bins, int min, int max);
void second_pass(bin_t *bin, int depth);
void third_pass(bin_t *bin);
void edge_third_pass(bin_t *bin);
void fast_third_pass(bin_t *bin);
void fast_sequence(char *str, int seq_no, int inv);
char *get_name(int n);
void check_bin_sizes();
int fast_half_overlap(char x[], char y[], int type_x, int type_y, int hook,
    int width, int full_flag, int *x0, int *xt, int *y0, int *yt, int *mis);
char *subseq(char *seq, int first, int last);
char inv(char c);
void fast_uf_union(int m, int n);
void fast_uf_destroy();
void fast_uf_init(int n);

```

cluster.c

```

/*=====
void main(int argc, char *argv[])
{
    int i;
    FILE *seqfile;

    printf ("Clustering program, by Compugen Ltd.\n");
    printf ("Command-line parameters:\n");
    printf ("size of mword: %d bits\n", sizeof (mword)*8);
    prologue(argc, argv, &seqfile);
    printf ("Checking data length...\n");
    fflush (stdout);
    check_data_length (seqfile);
    printf ("Found %d sequences, %d bases.\n", N_seq, N_base);

    printf ("Initializing Cyclic Queue\n");
    cq = Init_Cyclic_Queue();
    /* UF init (N_seq, &union_find); */
    fast_uf_init (N_seq);
    read_data_into_memory (N_base, N_seq, seqfile);
    printf ("Done.\n");
    fclose (seqfile);
    printf ("Checking bin sizes...\n");
    fflush (stdout);
    check_bin_sizes();
    printf ("Done.\n");
    printf ("Clustering...\n");
    first_pass ();
    printf ("%dId %d Smith-Waterman's, %d Unions\nEntered %d hooks\n",
        counter_sw, counter_merge, counter_cq);
    /* UF_destroy (union_find); */
    fast_uf_destroy ();
    print_sizes_in_CQ (cq, fp_hash);
    CQ_destroy (cq);
}

/*=====
void prologue (int argc, char *argv[], FILE **seqfile)
{
    char name[150], dir[100], fname[50], pairsname[50], tmp[50];
    int i;

    prm_argv (argc, argv, P_HELP,
        "dir = %s : directory", dir,
        "in = seq",
        "match = 10",
        "mismatch = -48",
        "gapop = 33",
        "gapext = 22",
        "cutoff = 40",
        "first = 0",
        "last = -1",
        "tuple = 9",
        "max_g = 3",
        "mem = 50",
        "depth = 3",
        "-sw ~",
        "-edge ~",
        "-only_full_overlap ~",
        "s : input file name", fname,
        "d : score for match", &match,
        "m : penalty for mismatch", &mismatch,
        "g : gap open penalty for short gaps", &gapop,
        "e : gap extend penalty for short gaps", &gapext,
        "c : cutoff plain, defines hills", &cutoff,
        "f : first bin to use", &first,
        "l : last bin to use (-1-unlimited)", &last,
        "t : tuple size", &tuple_size,
        "g : the largest gap", &max_dist,
        "m : memory size to use (Mbytes)", &mem_size,
        "d : no. of tuples to compare", &max_depth,
        "s : Do the band smith-waterman", &sw_flag,
        "e : cluster only near edges", &edge_flag,
        "o : only full overlap", &only_full_overlap);
}

```

cluster.c

A-252


```

        if (c == 'K') return 'M';
        if (c == 'M') return 'K';
        if (c == 'S') return 'S';
        if (c == 'W') return 'W';
        if (c == 'B') return 'V';
        if (c == 'D') return 'H';
        if (c == 'H') return 'D';
        if (c == 'V') return 'B';
        if (c == 'X') return 'X';
        return c;
    }

    /*=====*/
    void fast_uf_init (int n)
    {
        int i;

        if ((fast_uf_fathers = (int *) malloc (n * sizeof (int))) == NULL)
            error (1, -1, "Can't allocate fast_uf_fathers\n");
        for (i = 0; i < n; i++) fast_uf_fathers[i] = i;
        if ((fast_uf_brothers = (int *) malloc (n * sizeof (int))) == NULL)
            error (1, -1, "Can't allocate fast_uf_brothers\n");
        for (i = 0; i < n; i++) fast_uf_brothers[i] = i;
        if ((fast_uf_size = (int *) malloc (n * sizeof (int))) == NULL)
            error (1, -1, "Can't allocate fast_uf_size\n");
        for (i = 0; i < n; i++) fast_uf_size[i] = 1;
    }

    /*=====*/
    void fast_uf_destroy ()
    {
        free (fast_uf_fathers);
        free (fast_uf_brothers);
        free (fast_uf_size);
    }

    /*=====*/
    void fast_uf_union (int m, int n)
    {
        int i, j, small_size, big, small, tmp;

        /* Decide who's bigger (only the father is guaranteed to know) */
        if (fast_uf_size[fast_uf_fathers[m]] < fast_uf_size[fast_uf_fathers[n]]) {
            small = m;
            small_size = fast_uf_size[fast_uf_fathers[m]];
            big = n;
        } else {
            small = n;
            small_size = fast_uf_size[fast_uf_fathers[n]];
            big = m;
        }

        /* Hang the small cluster on the big one */
        for (i = small, j = 0; j < small_size; j++, i = fast_uf_brothers[i]) {
            fast_uf_fathers[i] = fast_uf_fathers[big];
        }

        /* Re-wire "brothers" */
        tmp = fast_uf_brothers[big];
        fast_uf_brothers[big] = fast_uf_brothers[small];
    }

```

cluster.c

```

        fast_uf_brothers[small] = tmp;

        /* Update size of father only */
        fast_uf_size[fast_uf_fathers[big]] += small_size;
    }

```

cluster.c

A-258

```

int i, loc;
if ( datap[est_no + 1] < datap[est_no] + pos + k )
    return 0;
*tuple = 0;
if ( inv == 0 ) {
    for ( i = 0; i < k; i++ ) {
        loc = datap[est_no] + pos + i;
        *tuple = *tuple ^ (GETDIR( data, loc) << 2*i);
    }
} else
    for ( i = 0; i < k; i++ ) {
        loc = datap[est_no] + pos + k - 1 - i;
        *tuple = *tuple ^ (GETDIR( data, loc) << 2*i);
    }
return 1;
}

/*=====*/
void word_to_chars( char *str, mword num, int len )
{
    int i;

    for ( i = 0; i < len; i++ ) {
        str[i] = init_string[num & 3];
        num = num >> 2;
    }

    /*=====*/
    void chars_to_word ( char *str, mword *num, int len, int inv)
    {
        int i;

        if ( inv == 0 ) {
            for ( i = *num = 0; i < len; i++ ) {
                if ( str[i] == 'N' )
                    *num ^= (random() & 3 << (2*i));
                else
                    *num ^= (conv_tab[str[i]] << (2*i));
            }
        } else {
            for ( i = *num = 0; i < len; i++ ) {
                if ( str[i] == 'N' )
                    *num = (*num << 2) ^ ((random () & 3) ^ 0x3);
                else
                    *num = (*num << 2) ^ (conv_tab[str[i]] ^ 0x3);
            }
        }

        /*=====*/
        void get_est( int est_no, char *str )
        {
            int loc, k;
            mword tmp = 0;

            k = datap[est_no + 1] - datap[est_no];
            if ( k > WDL ) {
                get_k_tuple( est_no, 0, 0, WDL - ((int)datap[est_no] & WDL), &tmp);
            }
        }
    }

```

comp_pack.c

```

loc = WDL - (datap[est_no] & WDL);
word_to_chars( str, tmp, loc);
for ( ; loc + WDL < k; loc += WDL )
    word_to_chars( str + loc, datap[est_no] + loc / WDL, WDL);
get_k_tuple( est_no, loc, 0, k - loc, &tmp);
word_to_chars( str + loc, tmp, k - loc);
} else {
    get_k_tuple( est_no, 0, 0, k, &tmp);
    word_to_chars( str, tmp, k);
}
str[k] = '\0';
}

/*=====*/
int get_comp_length (int seq_no)
{
    return (datap[seq_no + 1] - datap[seq_no]);
}

/*=====*/
int get_invariant_key (int seq_no, int loc, int tuple_size, unsigned short *inv)
{
    mword *word, key, bit;
    int pos, pointer;

    *inv = 0;
    pointer = datap[seq_no] + loc;
    if ( pointer + tuple_size > datap[seq_no+1] )
        return(-1);
    word = data + pointer / WDL;
    pos = pointer & (WDL - 1);
    key = (*word >> (2*pos));
    if ( pos > 0 ) key ^= (*word+1) << (2*(WDL - pos));
    key &= ((1 << 2*tuple_size) - 1);
    bit = (key >> tuple_size) & 1;
    if (bit) {
        key = invert_key (key, tuple_size);
        *inv = 1;
    }
    key = (key & ((1 << tuple_size) - 1)) ^
        ((key >> (tuple_size+1)) << tuple_size);
    return (key);
}

/*=====*/
int get_comp_key (int seq_no, int loc, int tuple_size)
{
    mword *word, key;
    int pos, pointer;

    pointer = datap[seq_no] + loc;
    if ( pointer + tuple_size > datap[seq_no+1] )
        return(-1);
    word = data + pointer / WDL;
    pos = pointer & (WDL - 1);
    key = (*word >> (2*pos));
    if ( pos > 0 ) key ^= (*word+1) << (2*(WDL - pos));
    key &= ((1 << 2*tuple_size) - 1);
    return (key);
}

```

comp_pack.c

```
#include <stdio.h>
#include <math.h>
#include "lib.h"
#include <string.h>
#include "cluster.h"
#include "comp_pack.h"

#define MAX_NAME_LENGTH 25
#define WDL (sizeof(mword)*4)
#define WBL (sizeof(mword)*8)
#define WL (sizeof(mword))

#define PUTDIT(a,i,d) ( a[i/WDL] = (a[i/WDL] & ~(3<<(2*(i%WDL)))) ^ (d<<(2*(i%WDL)
L)))

#define GETDIT(a,i) ( 3 & (a[i/WDL] >> (2*(i%WDL))) )

extern int conv_tab[256];
static char init_string[4];
static mword *data;
static int *datap;
static char *names, *types;

/*=====*/
void compress_and_add(seq_t *seq, int current_place)
{
    int i, j = current_place;

    for (i=0; i < seq->len; i++, j++) {
        PUTDIT(data, j, (mword)conv_tab[seq->d[i]]);
    }
}

/*=====*/
void init_alphabet(char *alphabet)
{
    int i;

    initialize(alphabet);
    for (i = 0; i < 4; i++)
        init_string[i] = alphabet[i];
}

/*=====*/
void read_data_into_memory (int m, int n, FILE *fp)
{
    seq_t *seq = new_seq(1000);
    int seq_num = 0, current_pointer = 0, total_mem;
    int i, flag;

    rewind(fp);
    data = (mword *)malloc( (m / WDL + 1) * WL );
    datap = (int *)malloc((n + 1) * sizeof( int ));
    names = (char *)malloc((n + 1) * MAX_NAME_LENGTH * sizeof( char ));
    types = (char *)malloc((n + 1) * sizeof( char ));
    memset((mword *)data, 0, m/WDL + 1);
    total_mem = (m / WDL + 1) * WL + (n + 1) * sizeof( int );
    printf("using %d bytes for reading data into memory\n", total_mem);
}
```

```
seq = read_seq( fp, seq, TRUE );
datap[seq_num] = current_pointer;
while ( seq != NULL ) {
    convert_n( seq );
    compress_and_add( seq, current_pointer );
    current_pointer += seq->len;
    /* Extract and save sequence name */
    for ( i = 0; seq->com[i] != ' '; i++)
        names[seq_num*MAX_NAME_LENGTH + i] = seq->com[i];
    /* extract and save sequence type */
    flag = 0;
    while (!flag) {
        for (;seq->com[i] != ' '; i++);
        if (seq->com[i+1] != 'T' || seq->com[i+2] != 'Y' || seq->com[i+3] != ' ')
            continue;
        flag = 1;
        for (i += 3; seq->com[i] != ' '; i++);
        if (seq->com[i] == 'E' && seq->com[i+1] == 'S' && seq->com[i+2] == 'T')
            types[seq_num] = EST;
        else
            if (seq->com[i] == 'R' && seq->com[i+1] == 'N' && seq->com[i+2] == 'A')
                types[seq_num] = RNA;
            else
                error (1, 0, "Came across an unknown sequence type, name = '%s'\n",
                    names[seq_num*MAX_NAME_LENGTH]);
    }

    seq = read_seq( fp, seq, TRUE );
    seq_num++;
    datap[seq_num] = current_pointer;
}

/*=====*/
void write_data_to_file (FILE *fp1, FILE *fp2, int m, int n)
{
    int len;

    fwrite (data, WL, (m / WDL + 1), fp1);
    len = n+1;
    fwrite (&len, 1, sizeof(int), fp2);
    fwrite (datap, len, sizeof( int), fp2);
    free (data);
}

/*=====*/
void read_data_pointers (FILE *fp)
{
    int len;
    fread (&len, 1, sizeof(int), fp);
    datap = (int *)malloc(len * sizeof( int ));
    fread (datap, len, sizeof( int), fp);
}

/*=====*/
int get_k_tuple (int est_no, int pos, int inv, int k, mword *tuple)
{
}
```



```
/*=====*/
mword invert_key (mword key, int tuple_size)
{
    static int first = 1, size, mask, s0, s1;
    static mword *tab;
    int i, j;

    if (!first) {
        first = 0;
        size = (tuple_size & 1) ? tuple_size + 1 : tuple_size;
        mask = (1 << size) - 1;
        s0 = 2*tuple_size - size;
        s1 = 2*size - 2*tuple_size;
        if (!((tab = (mword *) malloc ((1 << size) * sizeof (mword))))
            error (1, -1, "Can't allocate memory for tab in 'invert_key'.\n");
        for (i = 0; i < (1 << size); i++) {
            for (j = tab[i] = 0; j < size/2; j++)
                tab[i] = ((tab[i] << 2) ^ ((i >> (2*j)) & 3) ^ 3);
        }
    }
    return ((tab[key & mask] << s0) ^ (tab[key >> size] >> s1));
}

/*=====*/
char *get_name (int n)
{
    return (&names[MAX_NAME_LENGTH*n]);
}

/*=====*/
int get_type (int n)
{
    return ((int)types[n]);
}
```

```
#include <stdio.h>
#include "cyclic_queue.h"
```

```
static unsigned int hash_function(int loc, unsigned int est1, unsigned int est2)
{
    return (loc*loc*loc + est1 ^ est2) % CQ_SIZE;
}
```

```
CQ_struct *Init_Cyclic_Queue ()
{
    CQ_struct *tmp;
    int i;

    tmp = (CQ_struct *) malloc (sizeof(CQ_struct));
    if ( tmp == NULL )
        /*error (1, 0, "Can't allocate Cyclic Queue\n");*/
        printf ("Blahh\n");
    tmp->head = -1;
    for ( i = 0; i < CQ_SIZE; i++ ) {
        tmp->hash_table[i] = i;
        tmp->data[i].est1 = -1;
        tmp->data[i].est2 = -1;
        tmp->data[i].loc = -1;
        tmp->data[i].inv = 255;
        tmp->data[i].next = -1;
        tmp->data[i].cyc_loc = i;
        tmp->cyclic_array[i] = i;
        fflush (stdout);
    }
    return (tmp);
}
```

```
void CQ_destroy ( CQ_struct *cq )
{
    if ( cq != NULL )
        free (cq );
}
```

```
int find_in_CQ ( unsigned int est1, unsigned int est2,
                int loc, unsigned short inv, CQ_struct *cq )
{
    int i;
```

```
    i = hash_function ( loc, est1, est2 );
    if ( cq->hash_table[i] == -1 )
        return (-1);
    i = cq->hash_table[i];
    while ( cq->data[i].next != -1 && !(est1 == cq->data[i].est1 &&
                                     est2 == cq->data[i].est2 &&
                                     loc == cq->data[i].loc &&
                                     inv == cq->data[i].inv) )
```

```
    {
        i = cq->data[i].next;
    }
    if (est1 == cq->data[i].est1 &&
        est2 == cq->data[i].est2 &&
        loc == cq->data[i].loc &&
        inv == cq->data[i].inv ) {
        return(i);
    }
}
```

cyclic_queue.c

```
    else {
        return (-1);
    }
}

int insert_to_CQ ( unsigned int est1, unsigned int est2,
                  int loc, unsigned short inv, CQ_struct *cq )
{
    /* We assume that (est1, est2, loc) is not found in the queue */
    int tmp1, tmp2, tmp3, tmp;
```

```
    cq->head = (cq->head + 1) % CQ_SIZE;
    tmp = cq->cyclic_array[cq->head];
    if ( tmp < CQ_SIZE ) {
        tmp1 = cq->hash_table[tmp];
        tmp2 = cq->data[tmp1].next;
        cq->hash_table[tmp] = tmp2;
    } else {
        tmp1 = cq->data[tmp-CQ_SIZE].next;
        tmp2 = cq->data[tmp1].next;
        cq->data[tmp-CQ_SIZE].next = tmp2;
    }
    if ( tmp2 >= 0 )
        cq->cyclic_array[cq->data[tmp2].cyc_loc] = tmp;
    cq->data[tmp1].est1 = est1;
    cq->data[tmp1].est2 = est2;
    cq->data[tmp1].loc = loc;
    cq->data[tmp1].inv = inv;
    cq->data[tmp1].cyc_loc = cq->head;
    tmp3 = hash_function ( loc, est1, est2 );
    tmp2 = cq->hash_table[tmp3];
    cq->hash_table[tmp3] = tmp1;
    cq->cyclic_array[cq->head] = tmp3;
    if ( tmp2 == -1 ) {
        cq->data[tmp1].next = -1;
    }
    else {
        cq->data[tmp1].next = tmp2;
    }
    cq->cyclic_array[cq->data[tmp2].cyc_loc] = tmp1 + CQ_SIZE;
    return (cq->head);
}
```

```
void print_sizes_in_CQ ( CQ_struct *cq, FILE *fp )
{
    int i, tmp;
```

```
    int size;

    for ( i = 0; i < CQ_SIZE ; i++ ) {
        tmp = cq->hash_table[i];
        size = 0;
        while ( tmp != -1 ) {
            size++;
            tmp = cq->data[tmp].next;
        }
        fprintf (fp, "%d %d\n", i, size);
    }
}
```

cyclic_queue.c

A-263

Page

3

Sun Aug 9 10:40:38 1998

Listing for Adam Sartiell

cyclic_queue.c


```

#include <stdio.h>
#include "decide.h"

#define EST 0
#define RNA 1
#define MIN_STICK_LEN 45
#define FUZZ 5

extern int match, mismatch, gapop, gapext;
extern int local_flag;

int fast_half_overlap (char x[], char y[], int type_x, int type_y, int hook,
    int width, int full_flag, int *x0, int *xt, int *y0, int *yt, int *mis)
/*
 * Input:      input sequences.
 * x, y:      type_x, type_y: choose type of "decide" used.
 * x0, y0:    give hook
 * width:     of band
 * full_flag: flag. 1= look for full overlap only.
 *
 * Output:
 * x0, y0, xt, yt: coordinates of good alignment (if there is one).
 * mis:       number of mistakes in good alignment
 *
 * Return Value:
 * 0          Don't stick
 * 1          Stick
 */
{
    int type, i, j, rc, val;
    static int first = 1, decide[3][1000];
    int fuzz=FUZZ;

    if (local_flag) {
        fuzz = strlen(x);
    }

    /* Prepare the "decide" arrays. */
    if (first) {
        first = 0;
        for (i = 0; i < 3; i++)
            for (j = 0, val = -1; j < 1000; j++) {
                if (raw_decide[i][val+1] == j) val++;
                decide[i][j] = val;
            }
    }

    type = type_x + type_y;
    if (full_flag) {
        rc = serial_band_sw_ovr (x, strlen (x), Y, strlen (y), hook,
            match, mismatch, -gapop, -gapext, decide[type],
            MIN_STICK_LEN, fuzz, width, x0, y0, xt, yt, mis);
    } else {
        rc = serial_band_sw_two_directions_fa (x, strlen (x), Y, strlen (y), hook,
            match, mismatch, -gapop, -gapext, decide[type],
            MIN_STICK_LEN, fuzz, width, x0, y0, xt, yt, mis);
    }
    return (rc);
}

```

fast_band.c

fast_band.c

A-264

```

/* band_ovr.c
 * -----
 * banded sw routine with serial decide rejection that forces overlap.
 */

#include "/home/avi/include/usefull.h"

#define MAX_BAND 500
#define BIG_NEG (-999999999)

#define MAX_DECIDE 999

#define SWAP(x,y,t) {(t)=(x); (x)=(y); (y)=(t);}

/*=====
 * serial_band_sw_ovr(x,y,t) {(t)=(x); (x)=(y); (y)=(t);}
 *=====
 */
/*
 * decide_min_chk_len, start_band, band_width,
 * x_0,y_0,x_t,y_t,mis)
 */
/* inputs :
 * -----
 * x,y - the input sequences.
 * xlen,ylen - their lengths.
 * diag - the diagonal to band given in x-y units.
 * decide - decide[i] = maximal number of mistakes for alignment length i.
 * min_chk_len - minimal alignment length to begin serial rejection.
 * start_band - distance from start, from which an alignment can start.
 * band_width - we check from -band_width to +band_width around diag.
 * outputs:
 * -----
 * return code : 0/1 - stick/don't stick
 * in case of 1 :
 * x_0,y_0 - start of alignment.
 * x_t,y_t - end of alignment.
 * mis - number of mistakes in alignment.
 */
char *x,*y;
int xlen,ylen;
int diag;
int *decide;
int min_chk_len;
int start_band,band_width;
int match,mismatch,gap_open,gap_ext;
int *x_0,*y_0,*x_t,*y_t,*mis;
{
    int WorkArea0[25*MAX_BAND];
    int WorkArea1[25*MAX_BAND];

    int *M_score[2],*M_len[2],*M_start[2];
    int *D_score[2],*D_len[2],*D_start[2];
    int *I_score[2],*I_len[2],*I_start[2];

    int *temp;

```

```

int i,j;
int from_y,to_y,from,to;

int tmp_scr,xj;
int reject,nrejected;

int compare,add,len,nmis;

M_score[0] = &WorkArea0[MAX_BAND];
D_score[0] = &WorkArea0[3*MAX_BAND];
I_score[0] = &WorkArea0[5*MAX_BAND];
M_len[0] = &WorkArea0[13*MAX_BAND];
D_len[0] = &WorkArea0[15*MAX_BAND];
I_len[0] = &WorkArea0[17*MAX_BAND];
M_start[0] = &WorkArea0[19*MAX_BAND];
D_start[0] = &WorkArea0[21*MAX_BAND];
I_start[0] = &WorkArea0[23*MAX_BAND];
M_score[1] = &WorkArea1[MAX_BAND];
D_score[1] = &WorkArea1[3*MAX_BAND];
I_score[1] = &WorkArea1[5*MAX_BAND];
M_len[1] = &WorkArea1[13*MAX_BAND];
D_len[1] = &WorkArea1[15*MAX_BAND];
I_len[1] = &WorkArea1[17*MAX_BAND];
M_start[1] = &WorkArea1[19*MAX_BAND];
D_start[1] = &WorkArea1[21*MAX_BAND];
I_start[1] = &WorkArea1[23*MAX_BAND];

from_y = diag + band_width;
if (from_y >= 0)
    from_y = 0;
else
    from_y = -from_y;
to_y = xlen - 1 - diag + band_width;
if (to_y >= ylen)
    to_y = ylen-1;
if (to_y < 0)
    to_y = 0;

/*
 * printf("xlen=%d ylen=%d diag=%d from_y=%d to_y=%d\n",
 *        xlen,ylen,diag,from_y,to_y);
 */

/* return 0 if there's no chance of a long enough alignment */
if ((to_y - from_y + 1) < min_chk_len)
    return(0);
if ((to_y - from_y + 1) < MAX_DECIDE)
    reject = decide[to_y - from_y + 1] + 1;
else
    reject = decide[MAX_DECIDE];

/* init first column */
for (i=-band_width-1; i<=band_width+1; i++) {
    M_score[0][i] = D_score[0][i] = I_score[0][i] = BIG_NEG;
    M_len[0][i] = D_len[0][i] = I_len[0][i] = 0;

```

```

    }
    /* doing the alignment */
    /**
    print("diag=ed band_width=ed from_y=ed to_y=ed \n",
        diag, band_width, from_y, to_y);
    **/
    for (i=from_y; i<=to_y; i++) {
        from = MAX(-(i+diag), -band_width);
        /**
        printf("from=ed\n", from);
        **/
        M_score[1][-band_width-1] = BIG_NEG;
        M_len[1][-band_width-1] = 0;
        I_len[1][-band_width-1] = 0;
        M_score[0][band_width+1] = BIG_NEG;
        D_score[0][band_width+1] = BIG_NEG;
        for (j=-band_width; j<=-(i+diag); j++) {
            M_score[1][j] = D_score[1][j] = I_score[1][j] = BIG_NEG;
            M_len[1][j] = D_len[1][j] = I_len[1][j] = 0;
        }
        nrejected = 0;
        for (j=from; j<=band_width; j++) {
            xj = j + i + diag;
            /* low 16 bits of len are length, high 16 bits are number of
            mistakes */
            if (x[xj] == y[i]) {
                compare = 1;
                add = match;
            }
            else {
                compare = (1<<16) + 1;
                add = mismatch;
            }
            if ((x[xj] == 'N') || (y[i] == 'N')) {
                add = compare = 0;
            }
            /* match transitions */
            if ((M_score[0][j] >= I_score[0][j]) &&
                (M_score[0][j] >= D_score[0][j])) {
                M_score[1][j] = M_score[0][j] + add;
                M_start[1][j] = M_start[0][j];
                M_len[1][j] = M_len[0][j] + compare;
            }
            else {
                if (D_score[0][j] >= I_score[0][j]) {
                    M_score[1][j] = D_score[0][j] + gap_ext;
                    I_start[1][j] = I_start[0][j-1];
                    I_len[1][j] = I_len[0][j-1] + (1 + (1<<16));
                }
                else {
                    M_score[1][j] = M_score[1][j-1] + gap_open;
                    I_start[1][j] = M_start[1][j-1];
                    I_len[1][j] = M_len[1][j-1] + (1 + (1<<16));
                }
            }
            /* check if someone passed */
            len = M_len[1][j] & 0xffff;
            mmis = M_len[1][j] >> 16;
            if ((i>to_y-start_band) && (len >= min_chk_len)) {

```

```

if (nmis <= decide[len]) {
    *x_0 = (M_start[1][j] >> 16);
    *y_0 = (M_start[1][j] & 0xffff);
    *x_t = xj;
    *y_t = i;
    *mis = nmis;
    return(1);
}

)

if ((len >= min_chk_len) && (nmis >= reject))
    nrejected++;

/*
printf("i=%d j=%d xj=%d x[]=%lc y[]=%lc \n",
    i, xj, x[xj], y[i]);
printf("M :: scr=%d len=%d mis=%d start=%d \n",
    M_score[1][j], M_len[1][j] & 0xffff, M_len[1][j] >> 16,
    M_start[1][j] >> 16,
    M_start[1][j] & 0xffff);
printf("I :: scr=%d len=%d mis=%d start=%d \n",
    I_score[1][j], I_len[1][j] & 0xffff, I_len[1][j] >> 16,
    I_start[1][j] >> 16,
    I_start[1][j] & 0xffff);
printf("D :: scr=%d len=%d mis=%d start=%d \n",
    D_score[1][j], D_len[1][j] & 0xffff, D_len[1][j] >> 16,
    D_start[1][j] >> 16,
    D_start[1][j] & 0xffff);
**/
} /* end of j loop */

/* return 0 if all match states rejected */
if (nrejected == (band_width - from + 1))
    return(0);

/* swap all states */
SWAP(M_score[0], M_score[1], temp);
SWAP(M_len[0], M_len[1], temp);
SWAP(M_start[0], M_start[1], temp);
SWAP(D_score[0], D_score[1], temp);
SWAP(D_len[0], D_len[1], temp);
SWAP(D_start[0], D_start[1], temp);
SWAP(I_score[0], I_score[1], temp);
SWAP(I_len[0], I_len[1], temp);
SWAP(I_start[0], I_start[1], temp);

) /* end of i loop */

return(0);

/* return with nothing */

```



```

/*
 * bandsw.c
 * -----
 * banded sw routine with serial decide rejection.
 */

```

```

#include "/home/avi/include/usefull.h"

```

```

#define MAX_BAND 500
#define BIG_NEG (-99999999)

```

```

#define MAX_MISTAKES_PERCENT 6
#define MAX_BAND_WIDTH 250

```

```

#define MAX_DECIDE 999

```

```

#define SWAP(x,y,t) {(t)=(x); (x)=(y); (y)=(t);}

```

```

/*
 * serial_band_sw_two_directions(x,xlen,y,ylen,diag,
 * match,mismatch,gap_open,gap_ext,
 * decide,min_chk_len,start_band,band_width,
 * x_0,y_0,x_t,y_t,mis)
 */

```

```

/*
 * calls serial_band_sw and if it fails calls serial_band_sw_back
 */

```

```

char *x,*y;
int xlen,ylen;
int diag;
int *decide;
int min_chk_len;
int start_band,band_width;
int match,mismatch,gap_open,gap_ext;
int *x_0,*y_0,*x_t,*y_t,*mis;

```

```

{
    int rc;
    int from_y,to_y;
    int dlen,bwidth;

```

```

    if (from_y >= 0)
        from_y = 0;
    else
        from_y = -from_y;
    to_y = xlen - 1 - diag;
    if (to_y >= ylen)
        to_y = ylen - 1;
    if (to_y < 0)
        to_y = 0;
    dlen = to_y - from_y + 1;
    bwidth = (dlen*MAX_MISTAKES_PERCENT)/100;
    if (bwidth > band_width)
        band_width = bwidth;
    if (band_width > MAX_BAND_WIDTH)
        band_width = MAX_BAND_WIDTH;
}

```

fast_band_half.c

```

*/

```

```

rc = serial_band_sw(x,xlen,y,ylen,diag,match,mismatch,gap_open,gap_ext,
    decide,min_chk_len,start_band,band_width,
    x_0,y_0,x_t,y_t,mis);
if (rc)
    return(1);

```

```

rc = serial_band_sw_back(x,xlen,y,ylen,diag,match,mismatch,gap_open,gap_ext,
    decide,min_chk_len,start_band,band_width,
    x_0,y_0,x_t,y_t,mis);
return(rc);

```

```

}
/*
 * serial_band_sw_two_directions_fa(x,xlen,y,ylen,diag,
 * match,mismatch,gap_open,gap_ext,
 * decide,min_chk_len,start_band,band_width,
 * x_0,y_0,x_t,y_t,mis)
 */

```

```

/*
 * calls serial_band_sw and if it fails calls serial_band_sw_back
 */

```

```

char *x,*y;
int xlen,ylen;
int diag;
int *decide;
int min_chk_len;
int start_band,band_width;
int match,mismatch,gap_open,gap_ext;
int *x_0,*y_0,*x_t,*y_t,*mis;
{
    int rc;
    int from_y,to_y;

```

```

rc = serial_band_sw(x,xlen,y,ylen,diag,match,mismatch,gap_open,gap_ext,
    decide,min_chk_len,start_band,band_width,
    x_0,y_0,x_t,y_t,mis);
if (rc) {
    get_full_align_forward(x,xlen,y,ylen,diag,match,mismatch,gap_open,gap_ext,
        decide,min_chk_len,start_band,band_width,
        x_0,y_0,x_t,y_t,mis);
    return(1);
}

```

```

rc = serial_band_sw_back(x,xlen,y,ylen,diag,match,mismatch,gap_open,gap_ext,
    decide,min_chk_len,start_band,band_width,
    x_0,y_0,x_t,y_t,mis);

```

```

if (rc) {
    get_full_align_back(x,xlen,y,ylen,diag,match,mismatch,gap_open,gap_ext,
        decide,min_chk_len,start_band,band_width,
        x_0,y_0,x_t,y_t,mis);
    return(1);
}
return(rc);
}

```

fast_band_half.c


```

/*=====
serial_band_sw(x,xlen,y,ylen,diag,mismatch,gap_open,gap_ext,
               decide,min_chk_len,start_band,band_width,
               x_0,y_0,x_t,y_t,mis)
*/
/* inputs :
* -----
* x,y - the input sequences:
* xlen,ylen - their lengths.
* diag - the diagonal to band given in x-y units.
* decide - decide[i] = maximal number of mistakes for alignment length i.
* min_chk_len - minimal alignment length to begin serial rejection.
* start_band - distance from start, from which an alignment can start.
* band_width - we check from -band_width to +band_width around diag.
*
* outputs:
* -----
* return code : 0/1 - stick/don't stick
* in case of 1 :
* x_0,y_0 - start of alignment.
* x_t,y_t - end of alignment.
* mis - number of mistakes in alignment.
*/
char *x,*y;
int xlen,ylen;
int diag;
int *decide;
int min_chk_len;
int start_band,band_width;
int match,mismatch,gap_open,gap_ext;
int *x_0,*y_0,*x_t,*y_t,*mis;
{
    int WorkArea0[25*MAX_BAND];
    int WorkArea1[25*MAX_BAND];

    int *M_score[2],*M_len[2],*M_start[2];
    int *D_score[2],*D_len[2],*D_start[2];
    int *I_score[2],*I_len[2],*I_start[2];

    int *temp;

    int i,j;
    int from_y,to_y,from_to;

    int tmp_scr,xj;
    int reject,nrejected;

    int compare,add,len,nmis;

    int dlen,bwidth;

    M_score[0] = &WorkArea0[MAX_BAND];
    D_score[0] = &WorkArea0[3*MAX_BAND];
    I_score[0] = &WorkArea0[5*MAX_BAND];
    M_len[0] = &WorkArea0[13*MAX_BAND];
    D_len[0] = &WorkArea0[15*MAX_BAND];
    I_len[0] = &WorkArea0[17*MAX_BAND];

```

fast_band_half.c

```

M_start[0] = &WorkArea0[19*MAX_BAND];
D_start[0] = &WorkArea0[21*MAX_BAND];
I_start[0] = &WorkArea0[23*MAX_BAND];
M_score[1] = &WorkArea0[3*MAX_BAND];
D_score[1] = &WorkArea0[5*MAX_BAND];
I_score[1] = &WorkArea0[13*MAX_BAND];
M_len[1] = &WorkArea0[15*MAX_BAND];
D_len[1] = &WorkArea0[17*MAX_BAND];
I_len[1] = &WorkArea0[19*MAX_BAND];
M_start[1] = &WorkArea0[21*MAX_BAND];
D_start[1] = &WorkArea0[23*MAX_BAND];
I_start[1] = &WorkArea0[25*MAX_BAND];

    from_y = diag + band_width;
    if (from_y >= 0)
        from_y = 0;
    else
        from_y = -from_y;
    to_y = xlen - 1 - diag + band_width;
    if (to_y >= ylen)
        to_y = ylen - 1;
    if (to_y < 0)
        to_y = 0;

    /*
    printf("xlen=%d ylen=%d diag=%d from_y=%d to_y=%d\n",
           xlen,ylen,diag,from_y,to_y);
    */

    /* return 0 if there's no chance of a long enough alignment */
    if ((to_y - from_y + 1) < min_chk_len)
        return(0);

    if ((to_y - from_y + 1) < MAX_DECIDE)
        reject = decide(to_y - from_y + 1) + 1;
    else
        reject = decide(MAX_DECIDE);

    /* init first column */
    for (i=-band_width-1; i<=band_width+1; i++) {
        M_score[0][i] = D_score[0][i] = I_score[0][i] = BIG_NEG;
        M_len[0][i] = D_len[0][i] = I_len[0][i] = 0;
    }

    /* doing the alignment */

    /*
    printf("diag=%d band_width=%d from_y=%d to_y=%d\n",
           diag,band_width,from_y,to_y);
    */

    for (i=from_y; i<=to_y; i++) {
        from = MAX(-(i+diag),-band_width);

```

fast_band_half.c

```

/*
printf("from=%d\n", from);
**/

M_score[1][-band_width-1] = BIG_NEG;
I_score[1][-band_width-1] = BIG_NEG;
M_len[1][-band_width-1] = 0;
I_len[1][-band_width-1] = 0;
M_score[0][band_width+1] = BIG_NEG;
D_score[0][band_width+1] = BIG_NEG;

for (j=-band_width; j<=(i+diag); j++) {
    M_score[1][j] = D_score[1][j] = I_score[1][j] = BIG_NEG;
    M_len[1][j] = D_len[1][j] = I_len[1][j] = 0;
}

nrejected = 0;
for (j=from; j<=band_width; j++) {
    xj = j + i + diag;

    /* low 16 bits of len are length, high 16 bits are number of
       mistakes */
    if (x[xj] == y[i]) {
        compare = 1;
        add = match;
    } else {
        compare = (1<<16) + 1;
        add = mismatch;
    }

    if ((x[xj] == 'N') || (y[i] == 'N')) {
        add = compare = 0;
    }

    /* match transitions */
    if ((M_score[0][j] >= I_score[0][j]) &&
        (M_score[0][j] >= D_score[0][j])) {
        M_score[1][j] = M_score[0][j] + add;
        M_start[1][j] = M_start[0][j];
        M_len[1][j] = M_len[0][j] + compare;
    } else {
        if (D_score[0][j] >= I_score[0][j]) {
            M_score[1][j] = D_score[0][j] + add;
            M_start[1][j] = D_start[0][j];
            M_len[1][j] = D_len[0][j] + compare;
        } else {
            M_score[1][j] = I_score[0][j] + add;
            M_start[1][j] = I_start[0][j];

```

```

M_len[1][j] = I_len[0][j] + compare;
}

}

if ((i <= start_band) || (xj <= start_band)) {
    if (add > M_score[1][j]) {
        M_score[1][j] = add;
        M_start[1][j] = i; /* (xj<<16);
        M_len[1][j] = compare;
    }

}

/* deletions */
if (D_score[0][j+1] > M_score[0][j+1]) {
    D_score[1][j] = D_score[0][j+1] + gap_ext;
    D_start[1][j] = D_start[0][j+1];
    D_len[1][j] = D_len[0][j+1] + (1 + (1<<16));
} else {
    D_score[1][j] = M_score[0][j+1] + gap_open;
    D_start[1][j] = M_start[0][j+1];
    D_len[1][j] = M_len[0][j+1] + (1 + (1<<16));
}

/* insertions */
if (I_score[1][j-1] > M_score[1][j-1]) {
    I_score[1][j] = I_score[1][j-1] + gap_ext;
    I_start[1][j] = I_start[1][j-1];
    I_len[1][j] = I_len[1][j-1] + (1 + (1<<16));
} else {
    I_score[1][j] = M_score[1][j-1] + gap_open;
    I_start[1][j] = M_start[1][j-1];
    I_len[1][j] = M_len[1][j-1] + (1 + (1<<16));
}

/* check if someone passed */
len = M_len[1][j] & 0xffff;
nmis = M_len[1][j] >> 16;
if (len >= min_chk_len) {
    if (nmis <= decide[len]) {
        *x_0 = (M_start[1][j] >> 16);
        *y_0 = (M_start[1][j] & 0xffff);
        *x_t = xj;
        *y_t = i;
        *mis = nmis;
        return(1);
    }

    if (nmis >= reject)
        nrejected++;
}

```

```

/*
printf("i=%d j=%d xj=%d x[]=%lc y[]=%lc \n",
i,j,xj,x[j],y[i]);
printf("M :: scr=%d len=%d mis=%d start=%d \n",
M_score[1][j],M_len[1][j]&0xffff,M_len[1][j]>>16,
M_start[1][j]>>16,
M_start[1][j]&0xffff);
printf("I :: scr=%d len=%d mis=%d start=%d \n",
I_score[1][j],I_len[1][j]&0xffff,I_len[1][j]>>16,
I_start[1][j]>>16,
I_start[1][j]&0xffff);
printf("D :: scr=%d len=%d mis=%d start=%d \n",
D_score[1][j],D_len[1][j]&0xffff,D_len[1][j]>>16,
D_start[1][j]>>16,
D_start[1][j]&0xffff);
**/
) /* end of j loop */
/* return 0 if all match states rejected */
if (nrejected == (band_width - from + 1))
return(0);
/* swap all states */
SWAP(M_score[0],M_score[1],temp);
SWAP(M_len[0],M_len[1],temp);
SWAP(M_start[0],M_start[1],temp);
SWAP(D_score[0],D_score[1],temp);
SWAP(D_len[0],D_len[1],temp);
SWAP(D_start[0],D_start[1],temp);
SWAP(I_score[0],I_score[1],temp);
SWAP(I_len[0],I_len[1],temp);
SWAP(I_start[0],I_start[1],temp);
) /* end of i loop */

return(0);
/* return with nothing */
}
/*=====
serial_band_sw_back(x,xlen,y,ylen,diag,mismatch,gap_open,gap_ext,
decide,min_chk_len,start_band,band_width,
x_0,y_0,x_t,y_t,mis)
* inputs :
* -----
* x,y - the input sequences.
* xlen,ylen - their lengths.
* diag - the diagonal to band given in x-y units.
* decide - decide(i) = maximal number of mistakes for alignment length i.
* min_chk_len - minimal alignment length to begin serial rejection.
* start_band - distance from start, from which an alignment can start.
* band_width - we check from -band_width to +band_width around diag.
* outputs:

```

fast_band_half.c

fast_band_half.c

```

else
    from_y = -from_y;
    to_y = xlen - 1 - diag + band_width;
    if (to_y >= ylen)
        to_y = ylen-1;
    if (to_y < 0)
        to_y = 0;

/*
    printf("xlen=%d ylen=%d diag=%d from_y=%d to_y=%d\n",
           xlen, ylen, diag, from_y, to_y);
    **/

/* return 0 if there's no chance of a long enough alignment */
if ((to_y - from_y + 1) < min_chk_len)
    return(0);

if ((to_y - from_y + 1) < MAX_DECIDE)
    reject = decide(to_y - from_y + 1) + 1;
else
    reject = decide[MAX_DECIDE];

/* init first column */
for (i=-band_width-1; i<band_width+1; i++) {
    M_score[0][i] = D_score[0][i] = I_score[0][i] = BIG_NEG;
    M_len[0][i] = D_len[0][i] = I_len[0][i] = 0;
}

/* doing the alignment */

/*
    printf("diag=%d band_width=%d from_y=%d to_y=%d\n",
           diag, band_width, from_y, to_y);
    **/

for (i=from_y; i<=to_y; i++) {
    from = MAX(-(i+diag), -band_width);

    /*
        printf("from=%d\n", from);
        **/

    M_score[1][-band_width-1] = BIG_NEG;
    I_score[1][-band_width-1] = BIG_NEG;
    M_len[1][-band_width-1] = 0;
    I_len[1][-band_width-1] = 0;
    M_score[0][band_width+1] = BIG_NEG;
    D_score[0][band_width+1] = BIG_NEG;

    for (j=-band_width; j<=(i+diag); j++) {
        M_score[1][j] = D_score[1][j] = I_score[1][j] = BIG_NEG;
        M_len[1][j] = D_len[1][j] = I_len[1][j] = 0;

```

```

)

    nrejected = 0;
    for (j=from; j<=band_width; j++) {
        xj = xlen - 1 - (j + i + diag);
        yi = ylen - 1 - i;

        /* low 16 bits of len are length, high 16 bits are number of
           mistakes */

        if (x[xj] == y[yi]) {
            compare = 1;
            add = match;
        }
        else {
            compare = (1<<16) + 1;
            add = mismatch;
        }

        if ((x[xj] == 'N') || (y[yi] == 'N')) {
            add = compare = 0;
        }

        /* match transitions */

        if ((M_score[0][j] >= I_score[0][j]) &&
            (M_score[0][j] >= D_score[0][j])) {
            M_score[1][j] = M_score[0][j] + add;
            M_start[1][j] = M_start[0][j];
            M_len[1][j] = M_len[0][j] + compare;
        }
        else {
            if (D_score[0][j] >= I_score[0][j]) {
                M_score[1][j] = D_score[0][j] + add;
                M_start[1][j] = D_start[0][j];
                M_len[1][j] = D_len[0][j] + compare;
            }
            else {
                M_score[1][j] = I_score[0][j] + add;
                M_start[1][j] = I_start[0][j];
                M_len[1][j] = I_len[0][j] + compare;
            }
        }

        if ((i <= start_band) || ((xlen-1-xj) <= start_band)) {
            if (add >= M_score[1][j]) {
                M_score[1][j] = add;
                M_start[1][j] = yi ^ (xj<<16);
                M_len[1][j] = compare;
            }
        }
    }
}

```

```
/* deletions */
if (D_score[0][j+1] > M_score[0][j+1]) {
    D_score[1][j] = D_score[0][j+1] + gap_ext;
    D_start[1][j] = D_start[0][j+1];
    D_len[1][j] = D_len[0][j+1] + (1 + (1<<16));
} else {
    D_score[1][j] = M_score[0][j+1] + gap_open;
    D_start[1][j] = M_start[0][j+1];
    D_len[1][j] = M_len[0][j+1] + (1 + (1<<16));
}

/* insertions */
if (I_score[1][j-1] > M_score[1][j-1]) {
    I_score[1][j] = I_score[1][j-1] + gap_ext;
    I_start[1][j] = I_start[1][j-1];
    I_len[1][j] = I_len[1][j-1] + (1 + (1<<16));
} else {
    I_score[1][j] = M_score[1][j-1] + gap_open;
    I_start[1][j] = M_start[1][j-1];
    I_len[1][j] = M_len[1][j-1] + (1 + (1<<16));
}

/* check if someone passed */
len = M_len[1][j] & 0xffff;
nmis = M_len[1][j] >> 16;
if (len >= min_chk_len) {
    if (nmis <= decide[len]) {
        *x_t = (M_start[1][j] >> 16);
        *y_t = (M_start[1][j] & 0xffff);
        *x_0 = xj;
        *y_0 = yj;
        *mis = nmis;
        return(1);
    }
    if (nmis >= reject)
        nrejected++;
}

/*
printf("i=%d j=%d xj=%d x[]=%lc y[]=%lc \n",
        i,j,xj,x[xj],y[yj]);
printf("M :: scr=%d len=%d mis=%d start=%d \n",
        M_score[1][j],M_len[1][j],M_mis[1][j],M_start[1][j]>>16,
        M_start[1][j]&0xffff);
printf("I :: scr=%d len=%d mis=%d start=%d \n",
        I_score[1][j],I_len[1][j],I_mis[1][j],I_start[1][j]>>16,
        I_start[1][j]&0xffff);
printf("D :: scr=%d len=%d mis=%d start=%d \n",
        D_score[1][j],D_len[1][j],D_mis[1][j],D_start[1][j]>>16,
        D_start[1][j]&0xffff);
/**/
) /* end of j loop */
```

fast_band_half.c

```
/* return 0 if all match states rejected */
if (nrejected == (band_width - from + 1))
    return(0);
/* swap all states */
SWAP(M_score[0],M_score[1],temp);
SWAP(M_len[0],M_len[1],temp);
SWAP(M_start[0],M_start[1],temp);
SWAP(D_score[0],D_score[1],temp);
SWAP(D_len[0],D_len[1],temp);
SWAP(D_start[0],D_start[1],temp);
SWAP(I_score[0],I_score[1],temp);
SWAP(I_len[0],I_len[1],temp);
SWAP(I_start[0],I_start[1],temp);
) /* end of i loop */
return(0);
/* return with nothing */
}
```

fast_band_half.c

```

/*
 * bandsw.c
 * -----
 * banded sw routine with serial decide rejection.
 */

#include "/home/avi/include/usefull.h"

#define MAX_BAND 500
#define BIG_NEG (-999999999)

#define MAX_MISTAKES_PERCENT 6
#define MAX_BAND_WIDTH 250

#define MAX_DECIDE 999

#define SWAP(x,y,t) {(t)=(x); (x)=(y); (y)=(t);}

/*
 * =====
 * get_full_align_forward(x,xlen,y,ylen,diag,match,mismatch,gap_open,gap_ext,
 * decide,min_chk_len,start_band,band_width,
 * x_0,y_0,x_t,y_t,mis)
 */
/*
 * inputs :
 * -----
 * x,y - the input sequences.
 * xlen,ylen - their lengths.
 * diag - the diagonal to band given in x-y units.
 * decide - decide[i] = maximal number of mistakes for alignment length i.
 * min_chk_len - minimal alignment length to begin serial rejection.
 * start_band - distance from start, from which an alignment can start.
 * band_width - we check from -band_width to +band_width around diag.
 *
 * outputs:
 * -----
 * return code : 0/1 - stick/don't stick
 * in case of 1 :
 * x_0,y_0 - start of alignment.
 * x_t,y_t - end of alignment.
 * mis - number of mistakes in alignment.
 */
char *x,*y;
int xlen,ylen;
int diag;
int *decide;
int min_chk_len;
int start_band,band_width;
int match,mismatch,gap_open,gap_ext;
int *x_0,*y_0,*x_t,*y_t,*mis;
{
    int WorkArea0[25*MAX_BAND];
    int WorkArea1[25*MAX_BAND];

    int *M_score[2],*M_len[2],*M_start[2];
    int *D_score[2],*D_len[2],*D_start[2];
    int *I_score[2],*I_len[2],*I_start[2];

```

full-align.c

```

int *temp;

int i,j;
int from_y,to_y,from,to;

int tmp_scr,xj;
int reject,nrejected;

int compare,add,len,nmis;

int dlen,bwidth;
int best_score = BIG_NEG;

M_score[0] = &WorkArea0[MAX_BAND];
D_score[0] = &WorkArea0[3*MAX_BAND];
I_score[0] = &WorkArea0[5*MAX_BAND];
M_len[0] = &WorkArea0[13*MAX_BAND];
D_len[0] = &WorkArea0[15*MAX_BAND];
I_len[0] = &WorkArea0[17*MAX_BAND];
M_start[0] = &WorkArea0[19*MAX_BAND];
D_start[0] = &WorkArea0[21*MAX_BAND];
I_start[0] = &WorkArea0[23*MAX_BAND];
M_score[1] = &WorkArea1[MAX_BAND];
D_score[1] = &WorkArea1[3*MAX_BAND];
I_score[1] = &WorkArea1[5*MAX_BAND];
M_len[1] = &WorkArea1[13*MAX_BAND];
D_len[1] = &WorkArea1[15*MAX_BAND];
I_len[1] = &WorkArea1[17*MAX_BAND];
M_start[1] = &WorkArea1[19*MAX_BAND];
D_start[1] = &WorkArea1[21*MAX_BAND];
I_start[1] = &WorkArea1[23*MAX_BAND];

from_y = diag + band_width;
if (from_y >= 0)
    from_y = 0;
else
    from_y = -from_y;
to_y = xlen - 1 - diag + band_width;
if (to_y >= ylen)
    to_y = ylen-1;
if (to_y < 0)
    to_y = 0;

/*
 * printf("xlen=%d ylen=%d diag=%d from_y=%d to_y=%d\n",
 * **/
 * xlen,ylen,diag,from_y,to_y);
/*
 * return 0 if there's no chance of a long enough alignment */
if ((to_y - from_y + 1) < min_chk_len)
    return;
if ((to_y - from_y + 1) < MAX_DECIDE)
    reject = decide[to_y - from_y + 1] + 1;
else
    reject = decide[MAX_DECIDE];

```

full-align.c

A-274

```

/* init first column */
for (i=-band_width-1; i<=band_width+1; i++) {
    M_score[0][i] = D_score[0][i] = I_score[0][i] = BIG_NEG;
    M_len[0][i] = D_len[0][i] = I_len[0][i] = 0;
}

```

```

/* doing the alignment */

```

```

/*
printf("diag=%d band_width=%d from_y=%d to_y=%d \n",
    diag, band_width, from_y, to_y);
**/

```

```

for (i=from_y; i<=to_y; i++) {

```

```

    from = MAX(-(i+diag), -band_width);

```

```

    /*
    printf("from=%d\n", from);
    **/

```

```

    M_score[1][-band_width-1] = BIG_NEG;
    I_score[1][-band_width-1] = BIG_NEG;
    M_len[1][-band_width-1] = 0;
    I_len[1][-band_width-1] = 0;
    M_score[0][band_width+1] = BIG_NEG;
    D_score[0][band_width+1] = BIG_NEG;

```

```

    for (j=-band_width; j<=(i+diag); j++) {
        M_score[1][j] = D_score[1][j] = I_score[1][j] = BIG_NEG;
        M_len[1][j] = D_len[1][j] = I_len[1][j] = 0;
    }

```

```

    nrejected = 0;

```

```

    for (j=from; j<=band_width; j++) {

```

```

        xj = j + i + diag;

```

```

        /* low 16 bits of len are length, high 16 bits are number of
        mistakes */

```

```

        if (x[xj] == y[i]) {
            compare = 1;
            add = match;
        }

```

```

        else {
            compare = (1<<16) + 1;
            add = mismatch;
        }

```

```

        if ((x[xj] == 'N') || (y[i] == 'N')) {
            add = compare = 0;
        }

```

```

        /* match transitions */

```

full_align.c

```

    if ((M_score[0][j] >= I_score[0][j]) &&
        (M_score[0][j] >= D_score[0][j])) {

```

```

        M_score[1][j] = M_score[0][j] + add;
        M_start[1][j] = M_start[0][j];
        M_len[1][j] = M_len[0][j] + compare;
    } else {

```

```

        if (D_score[0][j] >= I_score[0][j]) {

```

```

            M_score[1][j] = D_score[0][j] + add;
            M_start[1][j] = D_start[0][j];
            M_len[1][j] = D_len[0][j] + compare;

```

```

        } else {

```

```

            M_score[1][j] = I_score[0][j] + add;
            M_start[1][j] = I_start[0][j];
            M_len[1][j] = I_len[0][j] + compare;

```

```

        }
    }

```

```

    if ((i <= start_band) || (xj <= start_band)) {

```

```

        if (add > M_score[1][j]) {

```

```

            M_score[1][j] = add;
            M_start[1][j] = i ^ (xj<<16);
            M_len[1][j] = compare;
        }
    }

```

```

    /* deletions */

```

```

    if (D_score[0][j+1] > M_score[0][j+1]) {
        D_score[1][j] = D_score[0][j+1] + gap_ext;
        D_start[1][j] = D_start[0][j+1];
        D_len[1][j] = D_len[0][j+1] + (1 + (1<<16));
    } else {
        D_score[1][j] = M_score[0][j+1] + gap_open;
        D_start[1][j] = M_start[0][j+1];
        D_len[1][j] = M_len[0][j+1] + (1 + (1<<16));
    }

```

```

    /* insertions */

```

```

    if (I_score[1][j-1] > M_score[1][j-1]) {
        I_score[1][j] = I_score[1][j-1] + gap_ext;
        I_start[1][j] = I_start[1][j-1];
        I_len[1][j] = I_len[1][j-1] + (1 + (1<<16));
    } else {
        I_score[1][j] = M_score[1][j-1] + gap_open;
        I_start[1][j] = M_start[1][j-1];
        I_len[1][j] = M_len[1][j-1] + (1 + (1<<16));
    }

```

full_align.c

```

/* check if someone passed */
len = M_len[1][j] & 0xffff;
nmis = M_len[1][j] >> 16;
if (len >= min_chk_len) {
    if ((M_score[1][j] > best_score) && (nmis <= decide(len))) {
        *x_0 = (M_start[1][j] >> 16);
        *y_0 = (M_start[1][j] & 0xffff);
        *x_c = xj;
        *y_t = i;
        *mis = nmis;
        best_score = M_score[1][j];
    }

    if (nmis >= reject)
        nrejected++;
}

/*
printf("i=%d j=%d xj=%d x[1]=%lc y[1]=%lc \n",
        i,j,xj,x[xj],y[i]);
printf("M :: scr=%d len=%d mis=%d start=%d,%d\n",
        M_score[1][j],M_len[1][j]&0xffff,M_len[1][j]>>16,
        M_start[1][j]>>16,
        M_start[1][j]&0xffff);
printf("I :: scr=%d len=%d mis=%d start=%d,%d\n",
        I_score[1][j],I_len[1][j]&0xffff,I_len[1][j]>>16,
        I_start[1][j]>>16,
        I_start[1][j]&0xffff);
printf("D :: scr=%d len=%d mis=%d start=%d,%d\n",
        D_score[1][j],D_len[1][j]&0xffff,D_len[1][j]>>16,
        D_start[1][j]>>16,
        D_start[1][j]&0xffff);
*/
} /* end of j loop */

/* return if all match states rejected */
if (nrejected == (band_width - from + 1))
    return;

/* swap all states */
SWAP(M_score[0],M_score[1],temp);
SWAP(M_len[0],M_len[1],temp);
SWAP(M_start[0],M_start[1],temp);
SWAP(D_score[0],D_score[1],temp);
SWAP(D_len[0],D_len[1],temp);
SWAP(D_start[0],D_start[1],temp);
SWAP(I_score[0],I_score[1],temp);
SWAP(I_len[0],I_len[1],temp);
SWAP(I_start[0],I_start[1],temp);

} /* end od i loop */
}
}
/*=====*/

```

```

/*=====*/
get_full_align_back(x,xlen,y,ylen,diag,match,mismatch,gap_open,gap_ext,
                    decide,min_chk_len,start_band,band_width,
                    x_0,y_0,x_t,y_t,mis)
/*
* inputs :
* -----
* * x,y - the input sequences.
* * xlen,ylen - their lengths.
* * diag - the diagonal to band given in x-y units.
* * decide - decide[i] = maximal number of mistakes for alignment length i.
* * min_chk_len - minimal alignment length to begin serial rejection.
* * start_band - distance from start, from which an alignment can start.
* * band_width - we check from -band_width to +band_width around diag.
*
* outputs:
* -----
* * return code : 0/1 - stick/don't stick
* * in case of 1 :
* *   x_0,y_0 - start of alignment.
* *   x_t,y_t - end of alignment.
* *   mis - number of mistakes in alignment.
*/
char *x,*y;
int xlen,ylen;
int diag;
int *decide;
int min_chk_len;
int start_band,band_width;
int match,mismatch,gap_open,gap_ext;
int *x_0,*y_0,*x_t,*y_t,*mis;
{
    int WorkArea0[25*MAX_BAND];
    int WorkArea1[25*MAX_BAND];

    int *M_score[2],*M_len[2],*M_start[2];
    int *D_score[2],*D_len[2],*D_start[2];
    int *I_score[2],*I_len[2],*I_start[2];

    int *temp;

    int i,j;
    int from_y,to_y,from,to;

    int tmp_scr,xj,yi;

    int reject,nrejected;

    int compare,add,len,nmis;
    int best_score = BIG_NEG;

    M_score[0] = &WorkArea0[MAX_BAND];
    D_score[0] = &WorkArea0[3*MAX_BAND];
    I_score[0] = &WorkArea0[5*MAX_BAND];
    M_len[0] = &WorkArea0[13*MAX_BAND];
    D_len[0] = &WorkArea0[15*MAX_BAND];
    I_len[0] = &WorkArea0[17*MAX_BAND];
    M_start[0] = &WorkArea0[19*MAX_BAND];

```



```

D_start[0] = &WorkArea0[21*MAX_BAND];
I_start[0] = &WorkArea0[23*MAX_BAND];
M_score[1] = &WorkArea0[MAX_BAND];
D_score[1] = &WorkArea0[3*MAX_BAND];
I_score[1] = &WorkArea0[5*MAX_BAND];
M_len[1] = &WorkArea0[13*MAX_BAND];
D_len[1] = &WorkArea0[15*MAX_BAND];
I_len[1] = &WorkArea0[17*MAX_BAND];
M_start[1] = &WorkArea0[19*MAX_BAND];
D_start[1] = &WorkArea0[21*MAX_BAND];
I_start[1] = &WorkArea0[23*MAX_BAND];

```

```
diag = xlen - ylen - diag;
```

```
from_y = diag + band_width;
```

```
if (from_y >= 0)
```

```
from_y = 0;
```

```
else
```

```
from_y = -from_y;
```

```
to_y = xlen - 1 - diag + band_width;
```

```
if (to_y >= ylen)
```

```
to_y = ylen-1;
```

```
if (to_y < 0)
```

```
to_y = 0;
```

```
/*
```

```
printf("xlen=%d ylen=%d diag=%d from_y=%d to_y=%d\n",
```

```
xlen,ylen,diag,from_y,to_y);
```

```
/**/
```

```
/* return if there's no chance of a long enough alignment */
```

```
if ((to_y - from_y + 1) < min_chk_len)
```

```
return;
```

```
if ((to_y - from_y + 1) < MAX_DECIDE)
```

```
reject = decide(to_y - from_y + 1) + 1;
```

```
else
```

```
reject = decide[MAX_DECIDE];
```

```
/* init first column */
```

```
for (i=band_width-1; i<band_width+1; i++) {
```

```
M_score[0][i] = D_score[0][i] = I_score[0][i] = BIG_NEG;
```

```
M_len[0][i] = D_len[0][i] = I_len[0][i] = 0;
```

```
}
```

```
/* doing the alignment */
```

```
/* printf("diag=%d band_width=%d from_y=%d to_y=%d\n",
```

```
diag,band_width,from_y,to_y);
```

```
/**/
```

full_align.c

```
for (i=from_y; i<to_y; i++) {
```

```
from = MAX(-(i+diag), -band_width);
```

```
/*
```

```
printf("from=%d \n", from);
```

```
/**/
```

```
M_score[1][band_width-1] = BIG_NEG;
```

```
I_score[1][band_width-1] = BIG_NEG;
```

```
M_len[1][band_width-1] = 0;
```

```
I_len[1][band_width-1] = 0;
```

```
M_score[0][band_width+1] = BIG_NEG;
```

```
D_score[0][band_width+1] = BIG_NEG;
```

```
for (j=-band_width; j<-(i+diag); j++) {
```

```
M_score[1][j] = D_score[1][j] = I_score[1][j] = BIG_NEG;
```

```
M_len[1][j] = D_len[1][j] = I_len[1][j] = 0;
```

```
}
```

```
nrejected = 0;
```

```
for (j=from; j<=band_width; j++) {
```

```
xj = xlen - 1 - (j + i + diag);
```

```
yj = ylen - 1 - i;
```

```
/* low 16 bits of len are length, high 16 bits are number of mistakes */
```

```
if (x[xj] == y[yj]) {
```

```
compare = 1;
```

```
add = match;
```

```
}
```

```
else {
```

```
compare = (1<<16) + 1;
```

```
add = mismatch;
```

```
}
```

```
if ((x[xj] == 'N') || (y[yj] == 'N')) {
```

```
add = compare = 0;
```

```
}
```

```
/* match transitions */
```

```
if (M_score[0][j] >= I_score[0][j]) &&
```

```
(M_score[0][j] >= D_score[0][j]) {
```

```
M_score[1][j] = M_score[0][j] + add;
```

```
M_start[1][j] = M_start[0][j];
```

```
M_len[1][j] = M_len[0][j] + compare;
```

```
} else {
```

```
if (D_score[0][j] >= I_score[0][j]) {
```

```
M_score[1][j] = D_score[0][j] + add;
```

```
M_start[1][j] = D_start[0][j];
```

```
M_len[1][j] = D_len[0][j] + compare;
```

full_align.c


```

    ) else (
        M_score[1][j] = I_score[0][j] + add;
        M_start[1][j] = I_start[0][j];
        M_len[1][j] = I_len[0][j] + compare;
    )
}

if ((i <= start_band) || ((xlen-1-x) <= start_band)) {
    if (add >= M_score[1][j]) {
        M_score[1][j] = add;
        M_start[1][j] = yi ^ (xj<<16);
        M_len[1][j] = compare;
    }
}

/* deletions */
if (D_score[0][j+1] > M_score[0][j+1]) {
    D_score[1][j] = D_score[0][j+1] + gap_ext;
    D_start[1][j] = D_start[0][j+1];
    D_len[1][j] = D_len[0][j+1] + (1 + (1<<16));
} else {
    D_score[1][j] = M_score[0][j+1] + gap_open;
    D_start[1][j] = M_start[0][j+1];
    D_len[1][j] = M_len[0][j+1] + (1 + (1<<16));
}

/* insertions */
if (I_score[1][j-1] > M_score[1][j-1]) {
    I_score[1][j] = I_score[1][j-1] + gap_ext;
    I_start[1][j] = I_start[1][j-1];
    I_len[1][j] = I_len[1][j-1] + (1 + (1<<16));
} else {
    I_score[1][j] = M_score[1][j-1] + gap_open;
    I_start[1][j] = M_start[1][j-1];
    I_len[1][j] = M_len[1][j-1] + (1 + (1<<16));
}

/* check if someone passed */
len = M_len[1][j] & 0xffff;
nmis = M_len[1][j] >> 16;
if (len >= min_chk_len) {
    if ((best_score < M_score[1][j]) && (nmis <= decide[len])) {
        *y_t = (M_start[1][j] >> 16);
        *y_c = (M_start[1][j] & 0xffff);
        *x_0 = xj;
        *mis = nmis;
        best_score = M_score[1][j];
    }
    if (nmis >= reject)

```

full_align.c

```

    nrejected++;
}

/*
printf("i=%d j=%d xj=%d x[]=%lc y[]=%lc \n",
    i,j,xj,x[xj],y[i]);
printf("M :: scr=%d len=%d mis=%d start=%d,%d\n",
    M_score[1][j],M_len[1][j],M_mis[1][j],M_start[1][j]>>16,
    M_start[1][j]&0xffff);
printf("I :: scr=%d len=%d mis=%d start=%d,%d\n",
    I_score[1][j],I_len[1][j],I_mis[1][j],I_start[1][j]>>16,
    I_start[1][j]&0xffff);
printf("D :: scr=%d len=%d mis=%d start=%d,%d\n",
    D_score[1][j],D_len[1][j],D_mis[1][j],D_start[1][j]>>16,
    D_start[1][j]&0xffff);
*/
} /* end of j loop */

/* return if all match states rejected */
if (nrejected == (band_width - from + 1))
    return;

/* swap all states */
SWAP(M_score[0],M_score[1],temp);
SWAP(M_len[0],M_len[1],temp);
SWAP(M_start[0],M_start[1],temp);
SWAP(D_score[0],D_score[1],temp);
SWAP(D_len[0],D_len[1],temp);
SWAP(D_start[0],D_start[1],temp);
SWAP(I_score[0],I_score[1],temp);
SWAP(I_len[0],I_len[1],temp);
SWAP(I_start[0],I_start[1],temp);
} /* end of i loop */

```

full_align.c

```

/*
 * Source file for hash package
 */

#include "cluster.h"
#include "hash_int.h"
#include <stdio.h>

/* Private hash function */
static unsigned hash_func (HS_int_struct *hs, mword key)
{
    return key % hs->size;
}

/* ===== */
HS_int_struct *init_int_hash (int size)
{
    HS_int_struct *tmp;
    int i;

    tmp = ( HS_int_struct * ) malloc (sizeof(HS_int_struct));
    if ( tmp == NULL )
        error(1, 0, "Can't allocate hash\n");
    tmp->size = size;
    tmp->hash = (el_int **) malloc(size * sizeof(el_int *));
    if ( tmp->hash == NULL )
        error(1, 0, "Can't allocate hash\n");
    for ( i = 0; i < size; i++)
        tmp->hash[i] = NULL;
    tmp->real_size = 0;
    return(tmp);
}

/* ===== */
void HS_int_destroy (HS_int_struct *hs)
{
    int i;
    el_int *elem, *nelem;

    if ( hs != NULL ) {
        if ( hs->hash != NULL ) {
            /* Free every element */
            for ( i = 0; i < hs->size; i++) {
                elem = hs->hash[i];
                while ( elem != NULL ) {
                    nelem = elem->next;
                    free(elem);
                    elem = nelem;
                }
            }
            /* Free hash table */
            free(hs->hash);
            free(hs);
        }
    }
}

```

hash_int.c

```

/* ===== */
void HS_int_insert (HS_int_struct *hs, mword key)
{
    el_int *the_elem;
    int ind;

    /* Create new element */
    the_elem = (el_int *)malloc(sizeof(el_int *));
    if ( the_elem == NULL ) {
        fprintf(stderr, "Can't allocate element\n");
        exit(-1);
    }
    the_elem->data = hs->real_size;
    hs->real_size++;
    the_elem->key = key;
    the_elem->next = NULL;

    /* Find hash ind */
    ind = hash_func(hs, key);

    /* If hash empty store element */
    if ( hs->hash[ind] == NULL )
        hs->hash[ind] = the_elem;
    else {
        the_elem->next = (hs->hash)[ind];
        hs->hash[ind] = the_elem;
    }
}

/* ===== */
int HS_int_find (HS_int_struct *hs, mword key)
{
    int ind;
    el_int *elem;

    /* Find hash index */
    ind = hash_func(hs, key);

    /* Find key */
    elem = hs->hash[ind];
    while ( (elem != NULL) && (elem->key != key) )
        elem = (el_int *)elem->next;
    if ( elem == NULL )
        return -1;
    else
        return elem->data;
}

/* ===== */
int HS_int_find_or_insert ( HS_int_struct *hs, mword key)
{
    int ind;

    ind = HS_int_find( hs, key);
    if ( ind == -1 ) {

```

hash_int.c

```

HS_int_insert( hs, key);
return (hs->real_size -1);
} else
return ind;
}

/*===== */
/* Does NOT print any title line. The function should print EXACTLY ONE item
per line using the printf format "%-30s%10d\n"
*/
void HASH_int_print_table (HS_int_struct *hs)
{
    el_int *elem;
    int i;

    for ( i = 0; i < hs->size; ++i ) {
        elem = hs->hash[i];
        while (elem != NULL ) {
            printf("%-30s%10d\n", elem->key, elem->data);
            elem = (el_int *)elem->next;
        }
    }
}

/*===== */
el_int *hash_next_element (HS_int_struct *hs, el_int *elem, int *i)
{
    el_int *tmp;

    if (elem == NULL) {
        /* If this is the first time - initialize tmp */
        tmp = hs->hash[0];
        (*i) = 0;
    } else
        /* Otherwise - advance to the next element */
        tmp = (el_int *)elem->next;

    /* If we are not at the end of a list finish */
    if (tmp != NULL)
        return(tmp);

    /* Search for the next non-empty hash entry */
    while (tmp == NULL && (*i) < hs->size) {
        (*i)++;
        tmp = hs->hash[*i];
    }

    /* Check if we are done */
    if ((*i) == hs->size ) return (NULL);
    return(tmp);
}

```

```

/*
 * Source file for hash package
 *
 *
#include "hash_st.h"
#include <stdio.h>

/* Private hash function */
static unsigned hash_func(HS_struct *hs, const char *key)
{
    int i;
    unsigned res;
    res = 0;
    for ( i = 0; i < strlen(key); ++i )
        res = (res * 101 + key[i]) ^ 0x5C;
    res = res % hs->size;
    return res;
}

HS_struct *init_hash (int size)
{
    HS_struct *tmp;
    tmp = ( HS_struct * ) malloc ( sizeof(HS_struct) );
    if ( tmp == NULL )
        error(1,0,"Can't allocate hash\n");
    tmp->size = size;
    tmp->hash = (el_char **)calloc(tmp->size,sizeof(el_char *));
    if ( tmp->hash == NULL )
        error(1,0,"Can't allocate hash\n");
    tmp->real_size = 0;
    return(tmp);
}

void HS_destroy (HS_struct *hs)
{
    int i;
    el_char *elem, *nelem;
    if ( hs != NULL ) {
        if ( hs->hash != NULL ) {
            /* Free every element */
            for ( i = 0; i < hs->size; ++i ) {
                elem = hs->hash[i];
                while ( elem != NULL ) {
                    nelem = (el_char *)elem->next;
                    free(elem->key);
                    free(elem);
                    elem = nelem;
                }
            }
        }
    }
}

```

hash_st.c

```

/* Free hash table */
}
free(hs->hash);
}
free(hs);
}

void HS_insert (HS_struct *hs, const char *key)
{
    el_char *elem;
    el_char *the_elem;
    int ind;

    /* Create new element */
    the_elem = (el_char *)malloc(sizeof(el_char *));
    if ( the_elem == NULL ) {
        fprintf(stderr,"Can't allocate element\n");
        exit(-1);
    }
    the_elem->data = hs->real_size;
    hs->real_size++;
    the_elem->key = (char *)malloc(sizeof(char) * strlen(key));
    if ( the_elem->key == NULL ) {
        fprintf(stderr,"Can't allocate key\n");
        exit(-1);
    }
    strcpy(the_elem->key,key);
    the_elem->next = NULL;
    /* Find hash ind */
    ind = hash_func(hs, key);
    /* If hash empty store element */
    if ( hs->hash[ind] == NULL )
        hs->hash[ind] = the_elem;
    else {
        the_elem->next = (struct el_char *)hs->hash[ind];
        hs->hash[ind] = the_elem;
    }
}

int HS_find (HS_struct *hs, const char *key)
{
    int ind;
    el_char *elem;

    /* Find hash index */
    ind = hash_func(hs, key);
    /* Find key */
    elem = hs->hash[ind];
    while ( (elem != NULL) && !(strcmp(elem->key,key)==0) )
        elem = (el_char *)elem->next;
}

```

hash_st.c

```

    if ( elem == NULL )
        return -1;
    else
        return elem->data;
}

int HS_find_or_insert ( HS_struct *hs, const char *key)
{
    int ind;

    ind = HS_find( hs, key);
    if ( ind == -1 ) {
        HS_insert( hs, key);
        return (hs->real_size -1);
    } else
        return ind;
}

/* Does NOT print any title line. The function should print EXACTLY ONE item
   per line using the printf format "%-30s%10d\n"
void HASH_print_table (HS_struct *hs)
{
    el_char *elem;
    int i;

    for ( i = 0; i < hs->size; ++i ) {
        elem = hs->hash[i];
        while (elem != NULL ) {
            printf("%-30s%10d\n", elem->key, elem->data);
            elem = (el_char *)elem->next;
        }
    }

    el_char *hash_st_next_element (HS_struct *hs, el_char *elem, int *i)
    {
        el_char *tmp;

        if (elem == NULL) {
            /* If this is the first time - initialize tmp */
            tmp = hs->hash[0];
            (*i) = 0;
        } else
            /* otherwise - advance to the next element */
            tmp = (el_char *)elem->next;

        /* If we are not at the end of a list finish */
        if (tmp != NULL)
            return(tmp);

        /* Search for the next non-empty hash entry */
        while (tmp == NULL && (*i) < hs->size) {
            (*i)++;
            tmp = hs->hash[*i];
        }

        /* Check if we are done */
        if ((*i) == hs->size ) return (NULL);
        return(tmp);
    }
}

```

hash_st.c

hash_st.c

```

#include <stdio.h>
#include <math.h>
#include <ctype.h>
#include <string.h>
#include <stdlib.h>
#include "lib.h"

int conv_tab[256];

void initialize(char *base_str)
{
    int i, j;
    for ( i = 0; i < 256; ++i )
        conv_tab[i] = strlen(base_str)-1;
    for ( j = 0; j < strlen(base_str); ++j )
        conv_tab[base_str[j]] = j;
}

seq_t *new_seq(int len)
{
    seq_t *seq;
    seq = (seq_t *)malloc(sizeof(seq_t));
    if ( seq == NULL )
        return NULL;
    seq->qual = 0;
    seq->len = MAX(len,1);
    seq->dlen = seq->len;
    seq->com = (char *)malloc(sizeof(char)*MAX_COM_SIZE);
    seq->d = (uchar *)malloc(sizeof(uchar)*seq->dlen);
    if ( seq->d == NULL || seq->com == NULL ) {
        if ( seq->d != NULL )
            free(seq->d);
        if ( seq->com != NULL )
            free(seq->com);
        free(seq);
        return NULL;
    }
    return seq;
}

void free_seq(seq_t *seq)
{
    free(seq->d);
    if ( seq->com != NULL ) {
        free(seq->com);
    }
    free(seq);
}

void print_seq(seq_t *seq, FILE *fp)
{
    #define PRINT_LINE 60
    int i;

```

lib.c

```

if ( fp == NULL )
    fp = stdout;
if ( seq->com == NULL )
    fprintf (fp, "> Segment of length %d\n", seq->len);
else if ( strlen(seq->com) )
    fprintf (fp, ">%s\n", seq->com);
for ( i = 0; i < seq->len; ++i ) {
    if ( (i != 0) && (!PRINT_LINE == 0) )
        fprintf (fp, "\n");
    fprintf (fp, "%c", seq->d[i]);
}
fprintf (fp, "\n");

void skip_whitespace(FILE *fp)
{
    int c;
    do {
        c = fgetc(fp);
    } while ( (c != EOF) && isspace(c) );
    if ( c != EOF )
        ungetc(c, fp);
}

seq_t *read_seq(FILE *fp, seq_t *seq, int hasCom)
{
    #define MAX_SEQ_SIZE 100000
    #define MARK '>'
    #define LINE_SIZE 1000

    int len, i, c;
    char com[MAX_COM_SIZE], *qptr;
    uchar line[LINE_SIZE];
    static uchar *seq_data;
    static int first_time = TRUE;
    if ( first_time ) {
        first_time = FALSE;
        seq_data = (uchar *)malloc(sizeof(uchar)*MAX_SEQ_SIZE);
        if ( seq_data == NULL ) {
            fprintf(stderr, "CANT ALLOCATE seq_data\n");
            return NULL;
        }
    }
    skip_whitespace(fp);
    /* Read Comment */
    if ( hasCom ) {
        c = fgetc(fp);
        if ( c == EOF )
            return NULL;

```

lib.c


```

if ( c != MARK ) {
    fprintf(stderr, "SEGMENT DOES NOT START WITH %c\n", MARK);
    return NULL;
}

fgets(com, MAX_COM_SIZE, fp);
if ( strlen(com) > 0 )
    com[strlen(com)-1] = 0;
}

/* Read segment */
len = 0;
while ( !feof(fp) ) {
    c = fgetc(fp);
    if ( c == EOF || c == MARK ) {
        if ( c == MARK )
            ungetc(c, fp);
        break;
    }
    ungetc(c, fp);
}

ungetc(c, fp);
fgets(line, LINE_SIZE, fp);
for ( i = 0; i < strlen(line); ++i )
    line[i] = toupper(line[i]);

if ( strlen(line) + len >= MAX_SEQ_SIZE ) {
    fprintf(stderr, "ERROR: segment is too large\n");
    return NULL;
}

memcpy(seq_data + len, line, strlen(line) - 1);
len += strlen(line) - 1;
}

if ( seq == NULL )
    seq = new_seq(len);
else if ( seq->d_len < len ) {
    free_seq(seq);
    seq = new_seq(len);
}

if ( seq == NULL ) {
    fprintf(stderr, "ERROR: can't allocate new segment\n");
    return NULL;
}

seq->len = len;
memcpy(seq->d, seq_data, len);

if ( seq->com != NULL ) {
    if ( hasCom ) {
        qptr = strstr(com, "BT=");
        if ( qptr == NULL ) {
            seq->qual = 0;
        } else {
            sscanf(qptr, "BT=%d", &seq->qual);
        }
    }
}

```

```

if ( seq->qual == 1 )
    seq->qual = -1;
else {
    qptr = strstr(com, "Q=");
    sscanf(qptr, "Q=%d", &seq->qual);
    if ( seq->qual > seq->len )
        seq->qual = seq->len;
}

strcpy(seq->com, com);
} else {
    seq->qual = 0;
    seq->com[0] = 0;
}
}

return seq;
}

void free_mif(mif_t *mif)
{
    free(mif->d);
    free(mif);
}

mif_t *new_mif(int base, int size)
{
    mif_t *mif;

    mif = (mif_t *)malloc(sizeof(mif_t));
    if ( mif == NULL )
        return NULL;

    mif->size = size;
    mif->base = base;
    mif->len = pow((double)base, (double)size);
    mif->tot = 0;

    mif->d = (int *)malloc(sizeof(int)*mif->len);
    if ( mif->d == NULL ) {
        free_mif(mif);
        return NULL;
    }
    memset(mif->d, 0, mif->len);

    return mif;
}

mif_t *read_mif(FILE *fp)
{
    mif_t *mif;
    int tot=0, count=0, d;

    mif = (mif_t *)malloc(sizeof(mif_t));
    if ( mif == NULL )
        return NULL;

    fscanf(fp, "base = %d\n", &mif->base);
    fscanf(fp, "size = %d\n", &mif->size);
    mif->len = pow((double)(mif->base), (double)(mif->size));
}

```

```

printf("Reading mifkad: len=%d, size=%d, base=%d\n",
       mif->len, mif->size, mif->base);

mif->d=(int *)malloc(mif->len*sizeof(int));
if ( mif->d == NULL ) {
    free_mif(mif);
    return NULL;
}

while (fscanf(fp, "%d\n", &d) != EOF)
{
    tot+=d;
    mif->d[count]=d;
    count++;
}
mif->tot=tot;
return(mif);
}

void write_mif(FILE *fp, mif_t *mif)
{
    int i;
    fprintf(fp, "base = %d\n", mif->base);
    fprintf(fp, "size = %d\n", mif->size);
    for ( i = 0; i < mif->len; ++i )
        fprintf(fp, "%d\n", mif->d[i]);
    fflush(fp);
}

void print_mif(mif_t *mif)
{
    int i;
    printf("Mifkad base=%d and size=%d : total length = %d\n",
           mif->base, mif->size, mif->len);
    for ( i = 0; i < mif->len; ++i )
        printf("#%5d = %d\n", i, mif->d[i]);
}

mif_t *calc_side_mif(mif_t *mif)
{
    mif_t *side_mif;
    int i, j, tmp;

    side_mif = new_mif(mif->base, mif->size - 1);
    if ( side_mif == NULL )
        return NULL;
    for ( i = 0; i < side_mif->len; i++) {
        tmp = 0;
        for ( j = 0; j < side_mif->base; j++)
            tmp += mif->d[i*mif->base + j];
        side_mif->d[i]=tmp;
    }
    /* print_mif(mif);

```

```

print_mif(side_mif); */
return(side_mif);
}

void create_start_weights(mif_t *mif, double *weights)
{
    int i;
    for ( i = 0; i < mif->len; ++i )
        weights[i] = MY_LOG( (mif->d[i] + 1) / (double) (mif->tot + mif->len) );
}

void create_next_weights(mif_t *mif, double *weights)
{
    int i;
    mif_t *side_mif;
    side_mif = calc_side_mif(mif);
    if ( side_mif == NULL )
        error(1, -1, "Can't allocate side mif");
    /* We must change this line to use side mifs : ADDITION (change) */
    for ( i = 0; i < mif->len; ++i )
        weights[i] = MY_LOG( (mif->d[i] + 1) /
                             (double) (side_mif->d[i*side_mif->len] + side_mif->base) );
    free_mif(side_mif);
}

int next_ind(int ind, uchar *s, int base, int size, int len)
{
    ind = (base*ind + conv_tab[*s]) % len;
    return ind;
}

int get_ind(uchar *s, int base, int size, int len)
{
    int ind, j;
    ind = 0;
    for ( j = 0; j < size; ++j ) {
        ind = base*ind + conv_tab[s[j]];
    }
    return ind;
}

void convert_n(seq_t *seq)
{
    int i;
    double x;
    for ( i = 0; i < seq->len; i++ ) {
        if ( seq->d[i] != 'A' && seq->d[i] != 'G' && seq->d[i] != 'T'
            && seq->d[i] != 'C' ) {
            x = drand48();
            if ( x < 0.25 )
                seq->d[i] = 'A';
            if ( x >= 0.25 && x < 0.5 )

```

Sun Aug 9 10:40:40 1998

Listing for Adam Sartiell

```
seq->d[i] = 'G';  
if ( x >= 0.5 && x < 0.75 )  
    seq->d[i] = 'C';  
if ( x >= 0.75 )  
    seq->d[i] = 'T';  
}  
}
```

```
#include <stdio.h>
#include <math.h>
#include "union_find.h"

unsigned int UF_find(UF_struct *uf, unsigned int ind)
{
    int ind1 = ind, tmp;

    /* Find root */
    while ( uf->fathers[ind] != ind )
        ind = uf->fathers[ind];

    /* path compression */
    while ( uf->fathers[ind1] != ind ) {
        tmp = ind1;
        ind1 = uf->fathers[ind1];
        uf->fathers[tmp] = ind;
    }

    return ind;
}

int UF_size(UF_struct *uf, unsigned int ind)
{
    unsigned int tmp;

    tmp = UF_find( uf, ind );
    return(uf->size[tmp]);
}

void UF_union(UF_struct *uf, unsigned int ind1, unsigned int ind2)
{
    unsigned int tmp, tmp1, tmp2;

    /* Get the roots */
    tmp1 = UF_find( uf, ind1 );
    tmp2 = UF_find( uf, ind2 );

    /* Find smallest depth */
    if ( uf->depth[tmp1] < uf->depth[tmp2] ) {
        tmp = tmp1;
        tmp1 = tmp2;
        tmp2 = tmp;
    }

    /* Fix */
    uf->fathers[tmp2] = tmp1;
    if ( uf->depth[tmp1] == uf->depth[tmp2] )
        uf->depth[tmp1]++;
    uf->size[tmp1] += uf->size[tmp2];
}

void UF_init( int size, UF_struct **uf )
{
    unsigned int i;

    *uf = (UF_struct *) malloc( sizeof( UF_struct ) );
    if ( *uf == NULL )
        error(1,0,"Can't allocate union-find structure");
}
```

union_find.c

```
(*uf)->fathers = (unsigned int *)malloc(sizeof(unsigned int)*size);
if ( (*uf)->fathers == NULL )
    error(1,0,"Can't allocate fathers");
for ( i = 0; i < (unsigned int)size; ++i )
    (*uf)->fathers[i] = i;

(*uf)->depth = (unsigned int *)malloc(sizeof(unsigned int)*size);
if ( (*uf)->depth == NULL )
    error(1,0,"Can't allocate depth");
memset((*uf)->depth, 0, size * sizeof(unsigned int));

(*uf)->size = (int *)malloc(sizeof(int)*size);
if ( (*uf)->size == NULL )
    error(1,0,"Can't allocate sizes array of union find");
for ( i = 0; i < size; i++)
    (*uf)->size[i] = 1;
}

void UF_destroy (UF_struct *uf)
{
    if ( uf == NULL )
        return;
    if ( uf->depth != NULL )
        free( uf->depth );
    if ( uf->fathers != NULL )
        free( uf->fathers );
    if ( uf->size != NULL )
        free( uf->size );
    free( uf );
}
```

union_find.c

A-287


```
#define BAD_BIN_NO 62
```

```
int bad_bins[BAD_BIN_NO][2] =
{
  {0x0000, 0}, /* AAAAAAAAAA */
  {0x0000, 1}, /* AAAAAAAAAA */
  {0x0224, 0}, /* ACACACACA */
  {0x0224, 1}, /* ACACACACA */
  {0x0488, 0}, /* AGAGAGAGA */
  {0x0488, 1}, /* AGAGAGAGA */
  {0x06cc, 0}, /* ATATATATA */
  {0x06cc, 1}, /* ATATATATA */
  {0x0811, 0}, /* CACACACAC */
  {0x0811, 1}, /* CACACACAC */
  {0x0ab5, 0}, /* CCCCCCCCC */
  {0x0ab5, 1}, /* CCCCCCCCC */
  {0x0cd9, 0}, /* CGCGCGCGC */
  {0x0cd9, 1}, /* CGCGCGCGC */
  {0x0efd, 0}, /* CTCTCTCTC */
  {0x0efd, 1}, /* CTCTCTCTC */
}
```

```

* 16 bad bins, for poly_X and poly_XY. They are folded to 8 bins by
* the "invariant" trick, but need to be for both directions so 16 again.
* To get the bin write the 9-tuple form 5' to 3'. For example CGCGCGCG.
* If the middle letter is G or T - use the opposite strand (in our case
* we use CGCGCGCG). Translate A->00, C->01, G->10, T->11 (we get
* 011001100110011001). Now drop the bit left of the middle, which is
* always a 0 (we get 1100110110011001) and translate to hex (0x0cd99).
* The bit in bad_bins[j][1] is the bit which tells if we flipped the
* 9-tuple. Since, in our case we need both CGCGCGCG and GCGCGCGG - we use
* both possibilities.
*/
```

```

{0x01082, 0}, /* AAGAAGAAG */
{0x01082, 1}, /* AAGAAGAAG */
{0x1b5f7, 0}, /* AGAAGAGA */
{0x1b5f7, 1}, /* AGAAGAGA */
{0x10420, 0}, /* GAAGAAGAA */
{0x10420, 1}, /* GAAGAAGAA */

{0x00408, 0}, /* AAAGAAAGA */
{0x00408, 1}, /* AAAGAAAGA */
{0x01020, 0}, /* AAGAAGAA */
{0x01020, 1}, /* AAGAAGAA */
{0x04080, 0}, /* AGAAGAAA */
{0x04080, 1}, /* AGAAGAAA */
{0x0ffd, 0}, /* GAAAGAAG */
{0x0ffd, 1}, /* GAAAGAAG */

{0x01428, 0}, /* AAGNAGGA */
{0x01428, 1}, /* AAGNAGGA */
{0x050a0, 0}, /* AGGAAGGA */
{0x050a0, 1}, /* AGGAAGGA */
{0x0fbf5, 0}, /* GGAAGGAG */
{0x0fbf5, 1}, /* GGAAGGAG */
{0x0bf7d, 0}, /* GAAGAAGG */
{0x0bf7d, 1}, /* GAAGAAGG */

{0x0fbd, 0}, /* AAGAGGAG */
{0x0fbd, 1}, /* AAGAGGAG */
{0x1bf77, 0}, /* AGAGGAAG */
{0x1bf77, 1}, /* AGAGGAAG */

```

bad_bins.h

```

{0x1bf77, 1}, /* AGAGGAGA */
{0x1422, 0}, /* GAGGAAGAG */
{0x11422, 1}, /* GAGGAAGAG */
{0x0508a, 0}, /* AGGAAGAGG */
{0x0508a, 1}, /* AGGAAGAGG */
{0x1aff5, 0}, /* GGAAGAGGA */
{0x1aff5, 1}, /* GGAAGAGGA */
{0x104a0, 0}, /* GAAGAGGAA */
{0x104a0, 1}, /* GAAGAGGAA */

{0x01482, 0}, /* AAGGAGAAG */
{0x01482, 1}, /* AAGGAGAAG */
{0x0bf7, 0}, /* AGGAGAAGG */
{0x0bf7, 1}, /* AGGAGAAGG */
{0x14428, 0}, /* GGAGAAGGA */
{0x14428, 1}, /* GGAGAAGGA */
{0x10a2, 0}, /* GAGAAGGAG */
{0x10a2, 1}, /* GAGAAGGAG */
{0x1bbf7, 0}, /* AGAAGGAGA */
{0x1bbf7, 1}, /* AGAAGGAGA */
{0xief7d, 0}, /* GAAGGAGAA */
{0xief7d, 1}, /* GAAGGAGAA */

```

bad_bins.h

*Using for Adam Santiel**Sun Aug 9 10:40:40 1998*

```
#define EST      0
#define RNA      1

typedef unsigned long mword;

typedef struct bin_t {
    struct bin_t *next;
    unsigned int  seq_no;
    unsigned short loc;
    unsigned short inv;
} bin_t;
```

cluster.h


```

#define WDL (sizeof(mword)*4)
#define WBL (sizeof(mword)*8)
#define WL (sizeof(mword))

#define PUTDIT(a,i,d) ( a[i/WDL] = (a[i/WDL] & ~(3<<(2*(i%WDL)))) ^ (d<<(2*(i%WDL))) )

#define GETDIT(a,i) ( 3 & (a[i/WDL] >> (2*(i%WDL))) )

void init_alphabet( char *alphabet );
void read_data_into_memory( int m, int n, FILE *fp );
int get_k_tuple( int est_no, int pos, int inv, int k, mword *tuple );
void word_to_chars( char *str, mword num, int len );
void get_est( int est_no, char *str );
int write_data_to_file( FILE *fp1, FILE *fp2, int m, int n );
int get_comp_length( int seq_no );
int get_comp_key( int seq_no, int loc, int tuple_size );
int get_invariant_key( int seq_no, int loc, int tuple_size, unsigned short *inv );
mword invert_key( mword key, int tuple_size );
char *get_name( int n );
int get_type( int n );

```



```
#define CQ_SIZE 2000000

typedef struct CQ_element {
    int next;
    unsigned int est1;
    unsigned int est2;
    int loc;
    unsigned short inv;
    int cyc_loc;
} CQ_element;

typedef struct CQ_struct {
    int cyclic_array[CQ_SIZE]; /* the cyclic array */
    int hash_table[CQ_SIZE]; /* The hash table array */
    CQ_element data[CQ_SIZE];
    int head;
} CQ_struct;

void CQ_destroy ( CQ_struct *cq);
CQ_struct *Init_Cyclic_Queue ();
int find_in_CQ ( unsigned int est1, unsigned int est2,
                int loc, unsigned short inv, CQ_struct *cq);
int insert_to_CQ ( unsigned int est1, unsigned int est2,
                int loc, unsigned short inv, CQ_struct *cq);
void print_sizes_in_CQ ( CQ_struct *cq, FILE *fp );
```


Sun Aug 9 10:40:41 1998

Using for Adam Sartiel

```
/*
 * Header file for hash package
 */

/* Macros */

#ifndef NULL
#define NULL 0
#endif

typedef struct el_int {
    mword key;
    unsigned int data;
    struct el_int *next;
} el_int;

typedef struct HS_int_struct {
    int real_size;
    int size;
    el_int **hash;
} HS_int_struct;

/* Function prototypes */

HS_int_struct *init_int_hash (int size);
void HS_int_destroy (HS_int_struct *hs);
void HS_int_insert (HS_int_struct *hs, mword key);
int HS_int_find (HS_int_struct *hs, mword key);
int HS_int_find_or_insert (HS_int_struct *hs, mword key);
void HASH_int_print_table (HS_int_struct *hs);
el_int *hash_next_element (HS_int_struct *hs, el_int *elem, int *i);
```

hash.h


```
/*
 * Header file for hash package
 */

/* Macros */

#ifndef NULL
#define NULL 0
#endif

typedef struct el_char {
    char *key;
    unsigned data;
    struct el_char *next;
} el_char;

typedef struct HS_struct {
    int real_size;
    int size;
    el_char **hash;
} HS_struct;

/* Function prototypes */

HS_struct *init_hash (int size);
void HS_destroy (HS_struct *hs);
void HS_insert (HS_struct *hs, const char *key);
int HS_find (HS_struct *hs, const char *key);
int HS_find_or_insert (HS_struct *hs, const char *key);
void HASH_print_table (HS_struct *hs);
el_char *hash_st_next_element (HS_struct *hs, el_char *elem, int *i);
```


Using for Adam Santel

Sun Aug 9 10:40:42 1998

```
typedef struct UF_struct {
    unsigned int *depth;
    unsigned int *fathers;
    int
} UF_struct;

int UF_size (UF_struct *uf, unsigned int ind);
unsigned int UF_find(UF_struct *uf, unsigned int ind);
void UF_union(UF_struct *uf, unsigned int ind1, unsigned int ind2);
void UF_init( int size, UF_struct **uf );
void UF_destroy (UF_struct *uf);
```

union find.h

Sun Aug 9 10:40:42 1998

Listing for Adam Sartiell

Makefile

Sun Aug 9 10:40:42 1998

Listing for Adam Sartiell

Makefile

```

CC = cc
OBJ = cluster.$(ARCH).o fast_band.$(ARCH).o hash_int.$(ARCH).o \
      comp_pack.$(ARCH).o fast_band_full.$(ARCH).o \
      hash_st.$(ARCH).o union_find.$(ARCH).o cyclic_queue.$(ARCH).o \
      fast_band_half.$(ARCH).o lib.$(ARCH).o \
      full_align.$(ARCH).o
SRC = cluster.c fast_band.c hash_int.c comp_pack.c fast_band_full.c \
      hash_st.c union_find.c cyclic_queue.c fast_band_half.c lib.c \
      full_align.c
INC = bad_bins.h cyclic_queue.h hash_st.h cluster.h decide.h lib.h \
      comp_pack.h hash_int.h union_find.h
CFLAGS = -c -O3 -I$(PRODR00T)/include
CL = cc
LFLAGS = -L$(PRODR00T)/lib/$(ARCH) -lm
LPRM = -lprm

all: cluster.$(ARCH)

cluster.$(ARCH): \
      cluster.$(ARCH).o \
      $(CL) $(OBJ) $(LFLAGS) $(LPRM) -o cluster.$(ARCH)

cluster.$(ARCH).o: \
      cluster.c $(INC) Makefile \
      $(CC) $(CFLAGS) cluster.c -o cluster.$(ARCH).o

fast_band.$(ARCH).o: \
      fast_band.c Makefile \
      $(CC) $(CFLAGS) fast_band.c -o fast_band.$(ARCH).o

full_align.$(ARCH).o: \
      full_align.c Makefile \
      $(CC) $(CFLAGS) full_align.c -o full_align.$(ARCH).o

fast_band_half.$(ARCH).o: \
      fast_band_half.c Makefile \
      $(CC) $(CFLAGS) fast_band_half.c -o fast_band_half.$(ARCH).o

fast_band_full.$(ARCH).o: \
      fast_band_full.c Makefile \
      $(CC) $(CFLAGS) fast_band_full.c -o fast_band_full.$(ARCH).o

hash_st.$(ARCH).o: \
      hash_st.c hash_st.h Makefile \
      $(CC) $(CFLAGS) hash_st.c -o hash_st.$(ARCH).o

hash_int.$(ARCH).o: \
      hash_int.c hash_int.h cluster.h Makefile \
      $(CC) $(CFLAGS) hash_int.c -o hash_int.$(ARCH).o

comp_pack.$(ARCH).o: \
      comp_pack.c comp_pack.h lib.h cluster.h Makefile \
      $(CC) $(CFLAGS) comp_pack.c -o comp_pack.$(ARCH).o

union_find.$(ARCH).o: \
      union_find.c union_find.h Makefile \
      $(CC) $(CFLAGS) union_find.c -o union_find.$(ARCH).o

lib.$(ARCH).o: \
      lib.c lib.h Makefile \
      $(CC) $(CFLAGS) lib.c -o lib.$(ARCH).o

cyclic_queue.$(ARCH).o: \
      cyclic_queue.c cyclic_queue.h Makefile \
      $(CC) $(CFLAGS) cyclic_queue.c -o cyclic_queue.$(ARCH).o

depend: $(SRCS)
      makedepend $(CFLAGS) $(SRCS)

clean:
      rm -f *.$(ARCH).o cluster.$(ARCH) core

install:
      cp cluster.$(ARCH) $(PRODR00T)/bin/$(ARCH)/cluster

# DO NOT DELETE THIS LINE -- make depend depends on it.

```



```
#include <stdio.h>
#include <malloc.h>
#include <math.h>
#include <string.h>
#include "/home/raveh/include/prm.h"
#include "structures.h"
```

```
#define MIN_LEN 10
#define LINE_SIZE 5000
#define MAX_SEQ_LEN 50000
```

```
#define N_TOKEN 9
char token[N_TOKEN][10] = {"VER", "ID", "CLUSTER", "LENGTH",
"DOC", "HEADER", "ALIGN", "FEAT", "///"};
int max_tr_no;
```

```
#define VER 0
#define ID 1
#define CLUSTER 2
#define LENGTH 3
#define DOC 4
#define HEADER 5
#define ALIGN 6
#define FEAT 7
#define CTG_END 8
```

```
#define SILENT 0
#define LOUD 1
#define OK 0
#define EST 0
#define RNA 1
#define TRS 2
```

```
int get_cluster_info (FILE *in, cluster_t **cluster);
int get_contig_info (FILE *in, contig_t *contig, int *EstInContig, int *ContigInClust, int *FoundClusterSize);
int get_next_line (FILE *in, char *line, int *type, int mode);
char *concat (char *seq1, char *seq2);
void print_cluster_info (cluster_t *cluster);
void calc_sizes (cluster_t *cluster);
void parse_header (cluster_t *cluster);
int look_for (char *pat, char *st, int start);
int make_clonetab (cluster_t *cluster, clone_t **clone);
void sort_contigs (cluster_t *cluster, clone_t *clonetab, int n_clone);
void output_cluster (FILE *out, cluster_t *cluster, clone_t *clone, int n_cl);
void print_sequence_data (FILE *out, seq_t *seq, char *c_name);
void print_doc (FILE *out, char *doc);
void find_cluster_size (char *line, int *EstInContig, int *ContigInClust, int *EstInClust, int *FoundClusterSize);
/*****
main (int argc, char *argv[])
{
    int i, j, rc, ok, n_clone;
    FILE *in, *out;
    char fname[100], oname[100], dir[100], name[100], line[LINE_SIZE];
    float version;
    cluster_t *cluster = NULL;
    clone_t *clonetab = NULL;
```

heirarchy.c

```
prn_argv (argc, argv, P_PRINT | P_HELP,
"dir" = "directory", dir,
"in" = "input file name", fname,
"out" = "output file name", oname,
"max_tr" = 100,
EOLIST);
```

```
sprintf (name, "%s/%s", dir, fname);
if ((in = fopen (name, "r")) == NULL)
error (1, -1, "Can't open input file '%s'\n", name);
sprintf (name, "%s/%s", dir, oname);
if ((out = fopen (name, "w")) == NULL)
error (1, -1, "Can't open output file '%s'\n", name);
/* fprintf (out, "VER\thierarchy 1.0\n"); */
```

```
/* Check first line for version number */
/* rc = (fgets (line, LINE_SIZE, in) == NULL);
if (rc == EOF) error (1, -1, "Empty input file.\n");
if (sscanf (line, "VER hierarchy %f", &version) != 1)
error (1, -1, "Bogus version information '%s'\n", line);
if (version != 1.0)
error (1, -1, "Unknown version %f.\n", version); */
```

```
do {
    rc = get_cluster_info (in, &cluster);
    calc_sizes (cluster);
    parse_header (cluster);
    n_clone = make_clonetab (cluster, &clonetab);
    sort_contigs (cluster, clonetab, n_clone);
    n_clone = make_clonetab (cluster, &clonetab); /* contig order has changed */
    calc_est (cluster); /* added by guy kol 17.5.98 */
    printf ("debug - in main cluster %s n_est=%d\n", cluster->name, cluster->found_n_est);
    output_cluster (out, cluster, clonetab, n_clone);
} while (rc == OK);
```

```
/* print_cluster_info (*cluster); */
close (in);
close (out);
```

```
int get_cluster_info (FILE *in, cluster_t **cluster)
{
    static int first = 1;
    static contig_t *contig;
    contig_t *prev = NULL;
    int rc;
    int EstInContig, ContigInClust, EstInClust, FoundClusterSize;
```

```
EstInContig = ContigInClust = EstInClust = 0;
destroy_cluster (cluster);
(*cluster) = create_cluster ();
if (first) {
    first = 0;
```

heirarchy.c

```

destroy_contig (&contig);
contig = create_contig ();
FoundClusterSize=0;
rc = get_contig_info (in, contig, &EstInContig, &ContigInClust, &EstInClust, &FoundClusterSize);
) else rc = OK;

strcpy ((*cluster)->name, contig->cluster);
while (!strcmp ((*cluster)->name, contig->cluster) && rc == OK) {
    contig->next = NULL;
    if (prev) prev->next = contig;
    else (*cluster)->contig = contig;
    prev = contig;
    contig = create_contig ();
    FoundClusterSize=0;
    rc = get_contig_info (in, contig, &EstInContig, &ContigInClust, &EstInClust, &FoundClusterSize);
}
/* We stop on the first contig in a new cluster, or on EOF */
return (rc);
}

/*=====*/
int get_contig_info (FILE *in, contig_t *contig, int *EstInContig, int *ContigInClust, int *EstInClust, int *FoundClusterSize)
{
    int rc, type, pos;
    int i, j;
    char line [LINE_SIZE], tmp [LINE_SIZE], tmp1 [LINE_SIZE], *tmp2, *first, *second, *third;
    seq_t *sequence, *prev_seq = NULL;
    align_t *alignment, *prev_al = NULL;
    feat_t *feature, *prev_feat = NULL;

    /* Get ID */
    rc = get_next_line (in, line, &type, LOUD);
    if (rc == EOF) return (EOF);
    if (type != ID) error (1, -1, "Missing ID line (%s).\n", line);
    sscanf (line, "%s %s", contig->name);

    /* Get CLUSTER */
    rc = get_next_line (in, line, &type, LOUD);
    if (rc == EOF || type != CLUSTER)
        error (1, -1, "Missing CLUSTER line (%s).\n", line);
    sscanf (line, "%s %s", contig->cluster);

    /* Get LENGTH */
    rc = get_next_line (in, line, &type, LOUD);
    if (rc == EOF || type != LENGTH)
        error (1, -1, "Missing LENGTH line (%s).\n", line);
    sscanf (line, "%s %d", &(contig->len));

    /* Get DOC */
    if (!contig->doc = (char *) malloc (1))
        error (1, -1, "Can't allocate documentation string\n");
    contig->doc[0] = '\0';

```

heirarchy.c

```

rc = get_next_line (in, line, &type, LOUD);
while (rc == OK && type == DOC) {
    for (pos = 1; pos < strlen(line) &&
        (line[pos] == ' ' || line[pos-1] != ' '); pos++);
    if (pos < strlen(line)) {
        strcpy (tmp1, line+pos);
        tmp2 = concat (contig->doc, tmp1);
        free (contig->doc);
        contig->doc = tmp2;
    }
    rc = get_next_line (in, line, &type, LOUD);
}

/* Get sequence data (in a loop) */
while (rc == OK && type == HEADER) {
    /* added by guy kol 17.5.98 */
    /* finding information about the cluster from rich fasta */
    /* its strange that it's in the contig but that's life */

    find_cluster_size (line, EstInContig, ContigInClust, EstInClust, FoundClusterSize);
    printf ("debug - information for contig %s ContigInClust: %d EstInClust: %d\n",
        contig->name, *ContigInClust, *EstInClust);

    contig->EstInContig = *EstInContig;
    contig->ContigInClust = *ContigInClust;
    contig->EstInClust = *EstInClust;

    sequence = create_sequence ();
    sequence->next = NULL;
    if (prev_seq) prev_seq->next = sequence;
    else contig->sequence = sequence;
    prev_seq = sequence;
    for (pos = 1; line[pos] == ' ' || line[pos-1] != ' '; pos++);
    sequence->header = strdup (line+pos);
    sscanf (sequence->header+1, "%s", sequence->name);

    /* Get ALIGN */
    rc = get_next_line (in, line, &type, LOUD);
    prev_al = NULL;
    while (rc == OK && type == ALIGN) {
        alignment = create_align ();
        alignment->next = NULL;
        if (prev_al) prev_al->next = alignment;
        else sequence->alignment = alignment;
        prev_al = alignment;
        sscanf (line, "%s (%d,%d)x(%d,%d)",
            &(alignment->a), &(alignment->b), &(alignment->c), &(alignment->d));
        rc = get_next_line (in, line, &type, LOUD);
    }

    /* Get FEAT */
    prev_feat = NULL;
    while (rc == OK && type == FEAT) {
        feature = create_feat ();
        feature->next = NULL;

```

heirarchy.c

```

if (prev_feat) prev_feat->next = feature;
else sequence->feature = feature;
prev_feat = feature;
for (pos = 1; line[pos] != ' ' || line[pos-1] != ' ' ; pos++) {
    strcpy (feature, line+pos);
    rc = get_next_line (in, line, &type, LOUD);
}
}
if (rc == EOF || type != CTG_END)
    error (1, -1, "Missing '///' line (%s).\n", line);
else return (OK);
}
/* ===== */
/* added by guy kol 17.5.98 */
/* finds the S2 tag and extracts information about number of contigs and ests */
/* in the cluster */

void find_cluster_size (char *line, int *EstInContig, int *ContigInClust, int *EstInClust, int *FoundClusterSize) {
    int j;
    char tmp[LINE_SIZE], *first, *second, *third;

    strcpy(tmp, line);
    /* printf ("debug looking on line %s\n", tmp); */
    j=0;

    /* Get num of contigs and est from text */

    j = look_for ("#S2 ", tmp, 0);
    if (j == -1) {
    }
    else {
        if (!(*FoundClusterSize)) {
            first= strtok(tmp+j+4, " ");
            sscanf(first, "%d", EstInContig);
            second= strtok((char *)NULL, " ");
            sscanf(second, "%d", ContigInClust);
            third= strtok((char *)NULL, " ");
            sscanf(third, "%d", EstInClust);
            printf ("debug - found EstInContig = %d ContigInClust = %d EstInClust = %d\n", *EstInContig, *ContigInClust, *EstInClust);
            *FoundClusterSize=1;
        }
    }
}

/* ===== */
int get_next_line (FILE *in, char *line, int *type, int mode)
/* Read next line from file and identify its type */
{
    int i, rc;

    rc = (fgets (line, LINE_SIZE, in) == NULL);
    if (rc) {
        if (mode == SILENT) return (EOF); /* There isn't another line in the file */
        else error (1, -1, "Unexpected EOF encountered.\n");
    }
}

```

heirarchy.c

```

}
for (i = 0; i < N_TOKEN; i++)
    if (!strcmp (token[i], line, strlen(token[i]))) {
        *type = i;
        break;
    }
if (i == N_TOKEN)
    error (1, -1, "Can't get type from '%s'. Stopping.\n", line);
return (OK);
}
/* ===== */
char *concat (char *seq1, char *seq2)
/* Concatenate two strings, cutting at newline, deleting trailing blanks */
{
    char *tmp;
    int i, len1, len2;

    for (len1 = 0; len1 < strlen (seq1) && seq1[len1] != '\n'; len1++);
    for (len2 = 0; len2 < strlen (seq2) && seq2[len2] != '\n'; len2++);
    while (seq1[len1-1] == ' ' && len1 > 0) len1--;
    while (seq2[len2-1] == ' ' && len2 > 0) len2--;
    if (! (tmp = (char *) malloc (len1 + len2 + 2)))
        error (1, -1, "concat: Can't allocate string\n");
    strcpy (tmp, seq1, len1);
    if (len1 > 0) {
        tmp[len1] = ' ';
        strcpy (tmp + len1 + 1, seq2, len2);
        tmp[len1+len2+1] = '\0';
    }
    else {
        strcpy (tmp + len1, seq2, len2);
        tmp[len1+len2] = '\0';
    }
    return (tmp);
}
/* ===== */
void calc_sizes (cluster_t *cluster)
{
    contig_t *contig;
    seq_t *seq;

    cluster->n_contig = cluster->n_seq = 0;
    for (contig = cluster->contig; contig != NULL; contig = contig->next) {
        cluster->n_contig++;
        contig->n_seq = 0;
        for (seq = contig->sequence; seq != NULL; seq = seq->next) {
            cluster->n_seq++;
            contig->n_seq++;
        }
    }
}
/* ===== */
void calc_est (cluster_t *cluster)
{
}

```

heirarchy.c


```

printf (tmp, "%s #TY %s #LN %d", seq->name, type, seq->len);
seq->header = concat (tmp, tmp2);
}
score = ((double)n5 - n3)/(contig->n_seq;
if (score > P_MINUS_Q) contig->type = 5;
else if (score < -1*P_MINUS_Q) contig->type = 3;
}
}
/*=====
int look_for (char *pat, char *st, int start)
{
    int i, j, flag;
    for (i = start; i < strlen (st); i++) {
        if (st[i] != pat[0]) continue;
        /* Check if pattern starts at place i */
        for (j = flag = 1; j < strlen (pat); j++) {
            if (st[i+j] != pat[j]) {
                flag = 0;
                continue;
            }
        }
        if (flag == 1) return (i);
    }
    return (-1);
}
/*=====
int make_clonetab (cluster_t *cluster, clone_t **clone)
{
    contig_t *contig;
    seq_t *seq;
    int i, j, k, n, c_no, s_no, cl_no;
    int dir1, dir2, tmp;
    clone_t *cl;
    if (*clone) free (*clone);
    (*clone) = create_clone_table (cluster->n_seq);
    c_no = cl_no = 0;
    for (contig = cluster->contig; contig != NULL; contig = contig->next) {
        s_no = 0;
        for (seq = contig->sequence; seq != NULL; seq = seq->next) {
            if (strlen (seq->clone) != 0) { /* seq has clone information */
                for (i = 0; i < cl_no; i++)
                    if (strcmp (seq->clone, (*clone)[i].name)) break;
                if (i == cl_no) {
                    strcpy ((*clone)[i].name, seq->clone);
                    cl_no++;
                }
                n = (*clone)[i].n;
                if (n < MAX_CLONE_MATES) {
                    (*clone)[i].contig[n] = c_no;
                    (*clone)[i].seq[n] = s_no;
                    (*clone)[i].dir[n] = seq->dir;
                    (*clone)[i].n++;
                } else
                    error (0, 0, "%d sequence of clone %s, cluster %s\n",

```

heirarchy.c

```

MAX_CLONE_MATES, (*clone)[i].name, cluster->name);
}
s_no++;
}
c_no++;
}
/* (Bubble) sort the clone table entries */
for (i = 0; i < cl_no; i++) {
    cl = &((*clone)[i]);
    for (j = 0; j < cl->n - 1; j++)
        for (k = j+1; k < cl->n; k++) {
            dir1 = cl->dir[j]; if (dir1) dir1 = 4; /* undirected seq */
            dir2 = cl->dir[k]; if (dir2) dir2 = 4; /* - in the middle */
            if (dir2 > dir1) { /* swap */
                tmp = cl->dir[k];
                cl->dir[k] = cl->dir[j];
                cl->dir[j] = tmp;
                tmp = cl->seq[k];
                cl->seq[k] = cl->seq[j];
                cl->seq[j] = tmp;
                tmp = cl->contig[k];
                cl->contig[k] = cl->contig[j];
                cl->contig[j] = tmp;
            }
        }
    return (cl_no);
}
/*=====
void sort_contigs (cluster_t *cluster, clone_t *clonetab, int n_clone)
/*
* This routine will usually work, but might result in a wrong order in
* complicated cases. It should be totally redesigned later!
*/
{
    int i, j, k, n, dir, old_dir, dir1, dir2, flag;
    int *typetab, tmp;
    contig_t **contab, *contig, *tmpp;
    if (!((contab) =
        (contig_t **) malloc (sizeof (contig_t *) * cluster->n_contig+1)))
        error(1, -1, "Can't allocate memory for contab\n");
    if (!((typetab) = (int *) malloc (sizeof (int) * cluster->n_contig+1)))
        error(1, -1, "Can't allocate memory for typetab\n");
    /* create pointer array, initialize directions to 0 */
    for (contig = cluster->contig, i = 0; contig != NULL;
        contig = contig->next, i++) {
        typetab[i] = contig;
        typetab[i] = 0;
    }
    /* Set directions according to clone-pairs */
    for (i = 0; i < n_clone; i++) {
        n = clonetab[i].n;
        /* If all the clone mates have the same direction (especially if there
        is only one of them) - don't use it */
        if (clonetab[i].dir[0] == clonetab[i].dir[n-1]) continue;

```

heirarchy.c

```

/* If all mates refer to the same contig - don't use it */
for (j = flag = 0; j < n; j++)
  if (clonetab[i].contig[j] != clonetab[i].contig[0]) flag = 1;
if (!flag) continue;

for (j = 0; j < n; j++) {
  dir = clonetab[i].dir[j];
  old_dir = typetab[clonetab[i].contig[j]];
  if (dir == old_dir || old_dir == 0)
    typetab[clonetab[i].contig[j]] = dir;
  else
    typetab[clonetab[i].contig[j]] = 0;
}

/* For undecided contigs - use contig type (calculated before) */
for (i = 0; i < cluster->n_contig; i++)
  if (typetab[i] == 0) {
    /* Can happen only when a sequence has clone but no direction info */
    /* fprintf(stderr, "Encountered an undirected contig %s.",
       contab[i]->name); */
    if (contab[i]->type != 0) {
      /* fprintf(stderr, "Calling it %d.\n", contab[i]->type); */
      typetab[i] = contab[i]->type;
    } else fprintf(stderr, "\n");
  }
}

/* (bubble) sort */
for (i = 0; i < cluster->n_contig-1; i++)
  for (j = i+1; j < cluster->n_contig; j++) {
    dir1 = typetab[i]; if (!dir1) dir1 = 4;
    dir2 = typetab[j]; if (!dir2) dir2 = 4;
    if (dir2 > dir1) {
      tmp = contab[i];
      contab[i] = contab[j];
      contab[j] = tmp;
      tmp = typetab[i];
      typetab[i] = typetab[j];
      typetab[j] = tmp;
    }
  }

/* update cluster pointers */
cluster->contig = contig = contab[0];
for (i = 0; i < cluster->n_contig-1; i++)
  contab[i]->next = contab[i+1];
contab[cluster->n_contig-1]->next = NULL;

free (contab);
free (typetab);
}

/*=====
void output_cluster (FILE *out, cluster_t *cluster, clone_t *clone, int n_cl)
{
  contig_t *contig;
  seq_t *seq;
  int i, j, k, c_no, s_no;

```

heirarchy.c

```

fprintf (out, "ID\t%s\n", cluster->name);
/* Missing: cluster documentation. To be added when Avi has it */

/* Printing of cluster size - added by guy kol 17.5.98 */

if ((cluster->found_n_contig == cluster->n_contig) && (cluster->found_n_est ==
cluster->n_est)) {
  fprintf (out, "DOC\t %d contigs and %d est in cluster\n", cluster->found_n_co
ntig, cluster->found_n_est);
}
else {
  fprintf (out, "DOC\t %d contigs and %d est in orig cluster but only %d conti
gs and %d est after assembly\n", cluster->found_n_contig, cluster->found_n_est, clu
ster->n_contig, cluster->n_est);
}

fprintf (out, "ENDDOC\n");
for (contig = cluster->contig; contig != NULL; contig = contig->next) {
  fprintf (out, "CONTIG\t%s\n", contig->name);
  fprintf (out, "LENGTH\t%d\n", contig->len);
  print_doc (out, contig->doc);
  fprintf (out, "ENDDOC\n");
}

/* Do the clones, one by one. First: transcripts */
for (i = 0; i < n_cl; i++) {
  for (j = 0; j < clone[i].n; j++) {
    c_no = clone[i].contig[j];
    s_no = clone[i].seq[j];
    for (k = 0, contig = cluster->contig; k < c_no; k++)
      contig = contig->next;
    for (k = 0, seq = contig->sequence; k < s_no; k++)
      seq = seq->next;

    if (seq->type == TRS) {
      if (j == 0) {
        fprintf (out, "TYPE\tTRS\n");
        fprintf (out, "CLONE\t%s\n", seq->clone);
      }
      print_sequence_data (out, seq, contig->name);
    }
  }
}

/* Standalone transcripts */
for (contig = cluster->contig; contig != NULL; contig = contig->next)
  for (seq = contig->sequence; seq != NULL; seq = seq->next)
    if (seq->clone[0] == '\0' && seq->type == TRS) {
      fprintf (out, "TYPE\tTRS\n");
      print_sequence_data (out, seq, contig->name);
    }
}

/* Next: RNAs */
for (i = 0; i < n_cl; i++) {
  for (j = 0; j < clone[i].n; j++) {
    c_no = clone[i].contig[j];
    s_no = clone[i].seq[j];
    for (k = 0, contig = cluster->contig; k < c_no; k++)

```

heirarchy.c


```

contig = contig->next;
for (k = 0, seq = contig->sequence; k < s_no; k++)
    seq = seq->next;

if (seq->type == RNA) {
    if (j == 0) {
        fprintf (out, "TYPE\trNA\n");
        fprintf (out, "CLONE\ts\n", seq->clone);
    }
    print_sequence_data (out, seq, contig->name);
}

}

/* Standalone RNAs */
for (contig = cluster->contig; contig != NULL; contig = contig->next)
    for (seq = contig->sequence; seq != NULL; seq = seq->next)
        if (seq->clone[0] == '\0' && seq->type == RNA) {
            fprintf (out, "TYPE\trNA\n");
            print_sequence_data (out, seq, contig->name);
        }

/* Finally: ESTs */
for (i = 0; i < n_cl; i++) {
    for (j = 0; j < clone[i].n; j++) {
        c_no = clone[i].contig[j];
        s_no = clone[i].seq[j];
        for (k = 0, contig = cluster->contig; k < c_no; k++)
            contig = contig->next;
        for (k = 0, seq = contig->sequence; k < s_no; k++)
            seq = seq->next;

        if (seq->type == EST) {
            if (j == 0) {
                fprintf (out, "TYPE\test\n");
                fprintf (out, "CLONE\ts\n", seq->clone);
            }
            print_sequence_data (out, seq, contig->name);
        }
    }
}

/* Standalone ESTs */
for (contig = cluster->contig; contig != NULL; contig = contig->next)
    for (seq = contig->sequence; seq != NULL; seq = seq->next)
        if (seq->clone[0] == '\0' && seq->type == EST) {
            fprintf (out, "TYPE\test\n");
            print_sequence_data (out, seq, contig->name);
        }

    fprintf (out, "SQ Sequence 5 BP:\n");
    fprintf (out, "      dummy\n");
    fprintf (out, "      /\n");
}

/*=====
void print_sequence_data (FILE *out, seq_t *seq, char *c_name)
{
    align_t *al;
    feat_t *feat;

    fprintf (out, "NAME\ts\n", seq->name);
}

```

heirarchy.c

```

fprintf (out, "WITH\ts\n", c_name);
if (seq->dir)
    fprintf (out, "ARROW\td\n", seq->dir);
else
    fprintf (out, "ARROW\tn\n");
for (al = seq->alignment; al != NULL; al = al->next)
    fprintf (out, "ALIGN\td\td\td\td\n", al->a, al->b, al->c, al->d);
for (feat = seq->feature; feat != NULL; feat = feat->next)
    fprintf (out, "FEAT\ts\n", feat->data);
print_doc (out, seq->header);
fprintf (out, "ENDDOC\n");
}

/*=====
void print_doc (FILE *out, char *doc)
/* Cut a long documentation line into several lines of no more than 72 chars */
{
    char line[73];
    int i, j, len, space, last, flag;

    i = j = flag = 0;
    len = strlen (doc);
    line[72] = '\0';

    if (strlen(doc) == 0) {
        fprintf (out, "DOC\t");
        return;
    }

    do {
        line[i] = doc[j];
        if (doc[j] == ' ') {
            space = j;
            last = i;
            i++; j++;
        }
        if (i == 72 || j == len) {
            if (j == len) flag = 1;
            if (flag) line[i] = '\0';
            else if (last) line[last] = '\0';
            else line[i] = '\0';
            j = space+1;
            i = 0;
            last = 0;
            if (line[strlen(line)-1] == '\n')
                line[strlen(line)-1] = '\0';
            fprintf (out, "DOC\ts\n", line);
        } while (!flag);
    }

/*
VER hierarchy 1.0 (one line in the whole file)
ID <cluster name>
DOC <cluster documentation line>
DOC <additional cluster documentation line> (prepared by avi for heirarchy)
ENDDOC
CONTIG <name of 1st contig>
LENGTH <length>
DOC <contig1 documentation line>

```

heirarchy.c


```

DOC      <contig1 additional documentation line>
ENDDOC
CONTIG
LENGTH  <name of 2nd contig>
DOC      <length>
DOC      <contig2 documentation line>
DOC      <contig2 additional documentation line>
DOC      <contig2 additional documentation line> (only 3 lines printed)
ENDDOC
TYPE     <RNA/EST>
CLONE    <name> (optional, only for TYPE=EST)
NAME     <name of sequence>
WITH     <contig name>
ARROW    <5/3/n>
ALIGN    (<start> <end>)x(<start> <end>)
ALIGN    (<start>,<end>)x(<start>,<end>)
FEAT     (<start>,<end>)x(<start>,<end>) (can be many of these)
FEAT     <type> (<start>,<end>) (type will be 3 letters from a predefined table)
FEAT     <type> (<start>,<end>)
FEAT     <type> (<start>,<end>) (can be many of these)
DOC      <documentation line for this EST/RNA>
DOC      <additional documentation line for this EST/RNA> (3 lines printed)
ENDDOC
NAME     ... (if in the same line)
TYPE     ... (next line)
//
ID        ... (next cluster)
*/

```

```
#include <stdio.h>
#include <malloc.h>
#include <math.h>
#include <string.h>
#include "structures.h"
```

```
cluster_t *create_cluster ()
{
    cluster_t *cluster;

    if (!((cluster) = (cluster_t *) malloc (sizeof (cluster_t))))
        error(1, -1, "Can't allocate memory for cluster\n");
    cluster->contig = NULL;
    cluster->n_contig = cluster->n_seq = 0;
    if (!((cluster->doc) = malloc (1)))
        error(1, -1, "Can't allocate memory for cluster->doc\n");
    cluster->doc[0] = '\0';
    return (cluster);
}
```

```
/*=====*/
void destroy_cluster (cluster_t **cluster)
{
    contig_t *tmp;
```

```
    if (*cluster) {
        if ((*cluster)->contig)
            do {
                tmp = (*cluster)->contig;
                (*cluster)->contig = tmp->next;
                destroy_contig (&tmp);
            } while ((*cluster)->contig);
            free (*cluster);
        *cluster = NULL;
    }
```

```
/*=====*/
contig_t *create_contig ()
{
    contig_t *contig;
```

```
    if (!((contig) = (contig_t *) malloc (sizeof (contig_t))))
        error(1, -1, "Can't allocate memory for contig\n");
    contig->sequence = NULL;
    contig->n_seq = contig->type = 0;
    return (contig);
}
```

```
/*=====*/
void destroy_contig (contig_t **contig)
{
    seq_t *tmp;
```

```
    if (*contig) {
        if ((*contig)->sequence)
            do {
                tmp = (*contig)->sequence;
                (*contig)->sequence = tmp->next;
                destroy_sequence (&tmp);
            }
```

structures.c

```
    } while ((*contig)->sequence);
    free (*contig);
    *contig = NULL;
}
```

```
/*=====*/
seq_t *create_sequence ()
{
    seq_t *sequence;
```

```
    if (!((sequence) = (seq_t *) malloc (sizeof (seq_t))))
        error(1, -1, "Can't allocate memory for sequence\n");
    sequence->header = NULL;
    sequence->alignment = NULL;
    sequence->feature = NULL;
    return (sequence);
}
```

```
/*=====*/
void destroy_sequence (seq_t **seq)
{
    align_t *tmpa;
    feat_t *tmpf;
```

```
    free ((*seq)->header);
    if ((*seq)->alignment)
        do {
            tmpa = (*seq)->alignment;
            (*seq)->alignment = tmpa->next;
            free (tmpa);
        } while ((*seq)->alignment);
    if ((*seq)->feature)
        do {
            tmpf = (*seq)->feature;
            (*seq)->feature = tmpf->next;
            free (tmpf);
        } while ((*seq)->feature);
    free (*seq);
}
```

```
/*=====*/
align_t *create_align ()
{
    align_t *align;
```

```
    if (!((align) = (align_t *) malloc (sizeof (align_t))))
        error(1, -1, "Can't allocate memory for align\n");
    return (align);
}
```

```
/*=====*/
feat_t *create_feat ()
{
    feat_t *feat;

    if (!((feat) = (feat_t *) malloc (sizeof (feat_t))))
        error(1, -1, "Can't allocate memory for feat\n");
    return (feat);
}
```

structures.c

```
/*=====*/
clone_t *create_clone_table (int n)
{
    clone_t *clonetab;
    int i;

    if (!((clonetab) = (clone_t *) malloc (sizeof (clone_t)*n+1)))
        error(1, -1, "Can't allocate memory for clone table\n");
    for (i = 0; i < n; i++)
        clonetab[i].n = 0;
    return (clonetab);
}

/*=====*/
void print_cluster_info(cluster_t cluster)
{
    contig_t *contig;
    seq_t *seq;

    printf ("CLUSTER\t%s (%d %d)\n", cluster.name,
            cluster.n_contig, cluster.n_seq);
    for (contig = cluster.contig; contig != NULL; contig = contig->next) {
        printf ("CONTIG\t%s,%d\t%d\t", contig->name, contig->ctype, contig->n_seq);
        for (seq = contig->sequence; seq != NULL; seq = seq->next)
            printf ("%s,%d\t", seq->clone, seq->dir);
        printf ("\n");
    }
    printf ("//\n");
}
```

```
#include <stdio.h>
#include <malloc.h>
#include <math.h>
#include <string.h>
#include "structures.h"
```

```
cluster_t *create_cluster ()
{
    cluster_t *cluster;

    if (!((cluster) = (cluster_t *) malloc (sizeof (cluster_t))))
        error(1, -1, "Can't allocate memory for cluster\n");
    cluster->contig = NULL;
    cluster->n_contig = cluster->n_seq = 0;
    cluster->found_n_contig=cluster->found_n_est=0;
    if (!((cluster->doc) = malloc (1)))
        error(1, -1, "Can't allocate memory for cluster->doc\n");
    cluster->doc[0] = '\0';
    return (cluster);
}
```

```
void destroy_cluster (cluster_t **cluster)
{
    contig_t *tmp;
```

```
    if (*cluster) {
        if ((*cluster)->contig)
            do {
                tmp = (*cluster)->contig;
                (*cluster)->contig = tmp->next;
                destroy_contig (&tmp);
            } while ((*cluster)->contig);
            free (*cluster);
        *cluster = NULL;
    }
```

```
contig_t *create_contig ()
{
    contig_t *contig;
```

```
    if (!((contig) = (contig_t *) malloc (sizeof (contig_t))))
        error(1, -1, "Can't allocate memory for contig\n");
    contig->sequence = NULL;
    contig->n_seq = contig->type = 0;
    contig->EstInClust=contig->ContInClust=contig->EstInClust=0;
    return (contig);
}
```

```
void destroy_contig (contig_t **contig)
{
    seq_t *tmp;
```

```
    if (*contig) {
        if ((*contig)->sequence)
            do {
                tmp = (*contig)->sequence;
```

structures.c

```
        (*contig)->sequence = tmp->next;
        destroy_sequence (&tmp);
    } while ((*contig)->sequence);
    free (*contig);
}
```

```
*contig = NULL;
```

```
seq_t *create_sequence ()
{
    seq_t *sequence;
```

```
    if (!((sequence) = (seq_t *) malloc (sizeof (seq_t))))
        error(1, -1, "Can't allocate memory for sequence\n");
    sequence->header = NULL;
    sequence->alignment = NULL;
    sequence->feature = NULL;
    return (sequence);
}
```

```
void destroy_sequence (seq_t **seq)
{
    align_t *tmpa;
```

```
    feat_t *tmpf;

    free ((*seq)->header);
    if ((*seq)->alignment)
        do {
```

```
            tmpa = (*seq)->alignment;
            (*seq)->alignment = tmpa->next;
            free (tmpa);
        } while ((*seq)->alignment);
    if ((*seq)->feature)
        do {
            tmpf = (*seq)->feature;
            (*seq)->feature = tmpf->next;
            free (tmpf);
        } while ((*seq)->feature);
    free (*seq);
}
```

```
align_t *create_align ()
{
    align_t *align;
```

```
    if (!((align) = (align_t *) malloc (sizeof (align_t))))
        error(1, -1, "Can't allocate memory for align\n");
    return (align);
}
```

```
feat_t *create_feat ()
{
    feat_t *feat;
```

```
    if (!((feat) = (feat_t *) malloc (sizeof (feat_t))))
        error(1, -1, "Can't allocate memory for feat\n");
}
```

structures.c

```

    return (feat);
}

/*=====*/
clone_t *create_clone_table (int n)
{
    clone_t *clonetab;
    int i;

    if (!((clonetab) = (clone_t *) malloc (sizeof (clone_t)*n+1)))
        error(1, -1, "Can't allocate memory for clone table\n");
    for (i = 0; i < n; i++)
        clonetab[i].n = 0;
    return (clonetab);
}

/*=====*/
void print_cluster_info(cluster_t cluster)
{
    contig_t *contig;
    seq_t *seq;

    printf ("CLUSTER\t%s (%d %d)\n", cluster.name,
            cluster.n_contig, cluster.n_seq);
    for (contig = cluster.contig; contig != NULL; contig = contig->next) {
        printf ("CONTIG\t%s, %d\t %d (", contig->name, contig->type, contig->n_seq);
        for (seq = contig->sequence; seq != NULL; seq = seq->next)
            printf ("%s, %d ", seq->clone, seq->dir);
        printf (")\n");
    }
    printf ("'\n");
}

```

```
#include <stdio.h>
#include <malloc.h>
#include <math.h>
#include <string.h>
#include "structures.h"
```

```
cluster_t *create_cluster ()
{
    cluster_t *cluster;

    if (!((cluster) = (cluster_t *) malloc (sizeof (cluster_t))))
        error(1, -1, "Can't allocate memory for cluster\n");
    cluster->contig = NULL;
    cluster->n_contig = cluster->n_seq = 0;
    if (!((cluster->doc) = malloc (1)))
        error(1, -1, "Can't allocate memory for cluster->doc\n");
    cluster->doc[0] = '\0';
    return (cluster);
}
```

```
/*=====*/
void destroy_cluster (cluster_t **cluster)
{
    contig_t *tmp;

```

```
    if (*cluster) {
        if ((*cluster)->contig)
            do {
                tmp = (*cluster)->contig;
                (*cluster)->contig = tmp->next;
                destroy_contig (&tmp);
            } while ((*cluster)->contig);
            free (*cluster);
        }
        *cluster = NULL;
    }
```

```
/*=====*/
contig_t *create_contig ()
{
    contig_t *contig;

```

```
    if (!((contig) = (contig_t *) malloc (sizeof (contig_t))))
        error(1, -1, "Can't allocate memory for contig\n");
    contig->sequence = NULL;
    contig->n_seq = contig->type = 0;
    return (contig);
}
```

```
/*=====*/
void destroy_contig (contig_t **contig)
{
    seq_t *tmp;

```

```
    if (*contig) {
        if ((*contig)->sequence)
            do {
                tmp = (*contig)->sequence;
                (*contig)->sequence = tmp->next;
                destroy_sequence (&tmp);
            }

```

structures.c

```
    } while ((*contig)->sequence);
    free (*contig);
    *contig = NULL;
}
```

```
/*=====*/
seq_t *create_sequence ()
{
    seq_t *sequence;

```

```
    if (!((sequence) = (seq_t *) malloc (sizeof (seq_t))))
        error(1, -1, "Can't allocate memory for sequence\n");
    sequence->header = NULL;
    sequence->alignment = NULL;
    sequence->feature = NULL;
    return (sequence);
}
```

```
/*=====*/
void destroy_sequence (seq_t **seq)
{
    align_t *tmpa;
    feat_t *tmpf;

```

```
    free ((*seq)->header);
    if ((*seq)->alignment)
        do {
            tmpa = (*seq)->alignment;
            (*seq)->alignment = tmpa->next;
            free (tmpa);
        } while ((*seq)->alignment);
    if ((*seq)->feature)
        do {
            tmpf = (*seq)->feature;
            (*seq)->feature = tmpf->next;
            free (tmpf);
        } while ((*seq)->feature);
    free (*seq);
}
```

```
/*=====*/
align_t *create_align ()
{
    align_t *align;

```

```
    if (!((align) = (align_t *) malloc (sizeof (align_t))))
        error(1, -1, "Can't allocate memory for align\n");
    return (align);
}
```

```
/*=====*/
feat_t *create_feat ()
{
    feat_t *feat;

```

```
    if (!((feat) = (feat_t *) malloc (sizeof (feat_t))))
        error(1, -1, "Can't allocate memory for feat\n");
    return (feat);
}
```

structures.c

```
/*=====*/
clone_t *create_clone_table (int n)
{
    clone_t *clonetab;
    int i;

    if (!((clonetab) = (clone_t *) malloc (sizeof (clone_t)*n+1)))
        error(1, -1, "can't allocate memory for clone table\n");
    for (i = 0; i < n; i++)
        clonetab[i].n = 0;
    return (clonetab);
}

/*=====*/
void print_cluster_info(cluster_t cluster)
{
    contig_t *contig;
    seq_t *seq;

    printf ("CLUSTER\t%s (%d %d)\n", cluster.name,
            cluster.n_contig, cluster.n_seq);
    for (contig = cluster.contig; contig != NULL; contig = contig->next) {
        printf ("CONTIG\t%s,%d\t %d ( ", contig->name, contig->type, contig->n_seq);
        for (seq = contig->sequence; seq != NULL; seq = seq->next)
            printf ("%s,%d ", seq->clone, seq->dir);
        printf (")\n");
    }
    printf ("///\n");
}
/*=====*/
```

Sun Aug 9 10:41:02 1998

Listing for Adam Santell

```

#define MAX_CLONE_MATES 100

typedef struct align_t {
    struct align_t *next;
    int a, b, c, d;
} align_t;

typedef struct feat_t {
    struct feat_t *next;
    char data[80];
} feat_t;

typedef struct seq_t {
    struct seq_t *next;
    char name[20], clone[40];
    int type, len, dir;
    char *header;
    align_t *alignment;
    feat_t *feature;
} seq_t;

typedef struct contig_t {
    struct contig_t *next;
    char name[20], cluster[20], *doc;
    int len, n_seq, type;
    seq_t *sequence;
    int EstInCont, ContInClust, EstInClust;
} contig_t;

typedef struct cluster_t {
    char name[20], *doc;
    int n_contig, n_seq, n_est;
    int found_n_contig, found_n_est;
    contig_t *contig;
} cluster_t;

typedef struct clone_t {
    char name[40];
    int contig[MAX_CLONE_MATES], seq[MAX_CLONE_MATES], dir[MAX_CLONE_MATES];
    int n;
} clone_t;

cluster_t *create_cluster ();
contig_t *create_contig ();
seq_t *create_sequence ();
align_t *create_align ();
feat_t *create_feat ();
clone_t *create_clone_table (int n);

void destroy_cluster (cluster_t **cluster);
void destroy_contig (contig_t **contig);
void destroy_sequence (seq_t **seq);
void print_cluster_info (cluster_t cluster);

```


Listing for Adam Sartiell Sun Aug 9 10:41:15 1998

```

/* * Source file for hash package
 * */
#include "hash_st.h"
#include <stdio.h>

/* Private hash function */
static unsigned hash_func(HS_struct *hs, const char *key)
{
    int i;
    unsigned res;
    res = 0;
    for ( i = 0; i < strlen(key); ++i )
        res = (res * 101 + key[i]) ^ 0x5C;
    res = res % hs->size;
    return res;
}

HS_struct *init_hash (int size)
{
    HS_struct *tmp;

    tmp = ( HS_struct * ) malloc ( sizeof(HS_struct));
    if ( tmp == NULL )
        error(1,0,"Can't allocate hash\n");
    tmp->size = size;
    tmp->hash = (el_char **)calloc(tmp->size, sizeof(el_char *));
    if ( tmp->hash == NULL )
        error(1,0,"Can't allocate hash\n");
    tmp->real_size = 0;
    return(tmp);
}

void HS_destroy (HS_struct *hs)
{
    int i;
    el_char *elem, *nelem;

    if ( hs != NULL ) {
        if ( hs->hash != NULL ) {
            /* Free every element */
            for ( i = 0; i < hs->size; ++i ) {
                elem = hs->hash[i];
                while ( elem != NULL ) {
                    nelem = (el_char *)elem->next;
                    free(elem->key);
                    free(elem);
                    elem = nelem;
                }
            }
        }
    }
}

```

hash_st.c

Listing for Adam Sartiell Sun Aug 9 10:41:15 1998

```

/* Free hash table */
free(hs->hash);
}
free(hs);
}

void HS_insert (HS_struct *hs, const char *key)
{
    el_char *elem;
    el_char *the_elem;
    int ind;

    /* Create new element */
    the_elem = (el_char *)malloc(sizeof(el_char *));
    if ( the_elem == NULL ) {
        fprintf(stderr, "Can't allocate element\n");
        exit(-1);
    }
    the_elem->data = hs->real_size;
    hs->real_size++;
    the_elem->key = (char *)malloc(sizeof(char) * strlen(key));
    if ( the_elem->key == NULL ) {
        fprintf(stderr, "Can't allocate key\n");
        exit(-1);
    }
    strcpy(the_elem->key, key);
    the_elem->next = NULL;
    /* Find hash ind */
    ind = hash_func(hs, key);
    /* If hash empty store element */
    if ( hs->hash[ind] == NULL )
        else {
            the_elem->next = (struct el_char *)hs->hash[ind];
            hs->hash[ind] = the_elem;
        }
}

int HS_find (HS_struct *hs, const char *key)
{
    int ind;
    el_char *elem;

    /* Find hash index */
    ind = hash_func(hs, key);
    /* Find key */
    elem = hs->hash[ind];
    while ( (elem != NULL) && !(strcmp(elem->key, key) == 0) )
        elem = (el_char *)elem->next;
}

```

hash_st.c

```

    if ( elem == NULL )
        return -1;
    else
        return elem->data;
}

int HS_find_or_insert ( HS_struct *hs, const char *key)
{
    int ind;

    ind = HS_find( hs, key);
    if ( ind == -1 ) {
        HS_insert( hs, key);
        return (hs->real_size -1);
    } else
        return ind;
}

/* Does NOT print any title line. The function should print EXACTLY ONE item
   per line using the printf format "%-30s%10d\n"
void HASH_print_table (HS_struct *hs)
{
    el_char *elem;
    int i;

    for ( i = 0; i < hs->size; ++i ) {
        elem = hs->hash[i];
        while (elem != NULL ) {
            printf("%-30s%10d\n", elem->key, elem->data);
            elem = (el_char *)elem->next;
        }
    }

    el_char *hash_st_next_element (HS_struct *hs, el_char *elem, int *i)
    {
        el_char *tmp;

        if (elem == NULL) {
            /* If this is the first time - initialize tmp */
            tmp = hs->hash[0];
            (*i) = 0;
        } else
            /* otherwise - advance to the next element */
            tmp = (el_char *)elem->next;

        /* If we are not at the end of a list finish */
        if (tmp != NULL)
            return(tmp);

        /* Search for the next non-empty hash entry */
        while (tmp == NULL && (*i) < hs->size) {
            (*i)++;
            tmp = hs->hash[*i];
        }

        /* Check if we are done */
        if ((*i) == hs->size ) return (NULL);
        return(tmp);
    }
}

```

```

    )

```

```
#include <stdio.h>
#include "prm.h"
#include "hash_st.h"
#include "union_find.h"

#define MAX_LINE_SIZE 30000
#define MEM_CONST 0.8

#define FOREVER 1
#define MAX_PARTNERS 4000
#define MAX_NAME_LEN 25

typedef struct point_t {
    unsigned int clust;
    int num_of_elements;
    unsigned int *elements;
} point_t;
```

```
/* global arguments */
UF_struct *uf;
FILE *fp, *seqfile, *ofp;
char *names;
int *cl_names;
short *dates;
int N_seq, pr;

/* function prototypes */
void prologue(int argc, char *argv[], FILE **seqfile);
void get_names_and_dates(FILE *seqfile);
void names_to_hash(HS_struct *hs);
void read_pairs(UF_struct *uf, HS_struct *hs, FILE *fp);
void print_sizes(UF_struct *uf);
void test_clustering(HS_struct *hs, UF_struct *uf, UF_struct *uf2);
void print_clusters(UF_struct *uf, HS_struct *hs);
void set_names(UF_struct *uf, HS_struct *hs);
/*=====*/
void main(int argc, char *argv[])
{
```

```
    FILE *seqfile;
    HS_struct *hs;
    UF_struct *uf;

    /* Get parameters and sequence names */
    printf ("Clustering comparison program, by Compugen Ltd.\n");
    printf ("Command-line parameters:\n");
    prologue(argc, argv, &seqfile);
    get_names_and_dates(seqfile);
    fclose(seqfile);

    /* Initialize hash tables and union-find structures */
    hs = init_hash(N_seq);
    names_to_hash(hs);
    UF_init(N_seq, &uf);
    read_pairs(uf, hs, fp);
    fclose(fp);

    /* Do the work */
```

make_names.c

```
set_names(uf, hs);
print_clusters(uf, hs);

/* Cleanup */
UF_destroy(uf);
HS_destroy(hs);
free(names);
free(dates);
fclose(ofp);
}

/*=====*/
void prologue(int argc, char *argv[], FILE **seqfile)
{
    char name[150], dir[100], fname[50], pname1[50], pname2[50];
    int i;

    prm_argv(argc, argv, P_PRINT | P_HELP,
             "dir = . %s ! directory", dir,
             "in = seq %s ! input file name", fname,
             EOLIST);

    sprintf(name, "%s/%s", dir, fname);
    if ((*seqfile = fopen(name, "r")) == NULL)
        error(1, -1, "Can't open sequence file %s\n", name);
    sprintf(name, "%s/%s.pairs", dir, fname);
    if ((fip = fopen(name, "r")) == NULL)
        error(1, -1, "Can't open pairfile %s\n", name);
    sprintf(name, "%s/%s.clu", dir, fname);
    if ((ofp = fopen(name, "w")) == NULL)
        error(1, -1, "Can't open outfile %s\n", name);

    /*=====*/
    void get_names_and_dates(FILE *seqfile)
    {
        int i, n, day, month, year;
        char st[MAX_LINE_SIZE], a, b, c;

        rewind(seqfile);
        N_seq = n = 0;

        do {
            if (fgets(st, MAX_LINE_SIZE, seqfile) == NULL)
                break;
            if (st[0] == '>') N_seq++;
        } while (FOREVER);

        printf ("Found %d sequences in input file\n", N_seq);
        rewind(seqfile);
        if (names = (char *)malloc(MAX_NAME_LEN*N_seq) == NULL)
            error(1, -1, "Can't allocate memory for names\n");
        bzero(names, MAX_NAME_LEN*N_seq);

        if (dates = (short *)malloc((sizeof(short))*N_seq) == NULL)
            error(1, -1, "Can't allocate memory for dates\n");
        bzero(dates, (sizeof(short))*N_seq);

        if (cl_names = (int *)malloc(sizeof(int)*N_seq) == NULL)
            error(1, -1, "Can't allocate memory for cl_names\n");
```

make_names.c

```

for (i = 0; i < N_seq; i++) cl_names[i] = -1;
do {
    if (fgets (st, MAX_LINE_SIZE, seqfile) == NULL)
        break;
    if (st[0] == '>') {
        for (i = 0; st[i+1] != '\n'; i++)
            names[MAX_NAME_LEN*n + i] = st[i+1];
        for (i = 0; st[i] != '\n'; i++)
            if (st[i+1] != 'D' || st[i+2] != 'T', i++)
                /* Now we have found the date record. Parse it. */
                day = (st[i+4] - '0')*10 + st[i+5] - '0';
                year = (st[i+11] - '0')*1000 + (st[i+12] - '0')*100 +
                    (st[i+13] - '0')*10 + st[i+14] - '0' - 1950;
                a = st[i+7];
                b = st[i+8];
                c = st[i+9];
                month = 0;
                if (a == 'J' && b == 'A' && c == 'N') month = 1;
                if (a == 'F' && b == 'E' && c == 'B') month = 2;
                if (a == 'M' && b == 'A' && c == 'R') month = 3;
                if (a == 'A' && b == 'P' && c == 'R') month = 4;
                if (a == 'M' && b == 'A' && c == 'Y') month = 5;
                if (a == 'J' && b == 'U' && c == 'N') month = 6;
                if (a == 'J' && b == 'U' && c == 'L') month = 7;
                if (a == 'A' && b == 'U' && c == 'G') month = 8;
                if (a == 'S' && b == 'E' && c == 'P') month = 9;
                if (a == 'O' && b == 'C' && c == 'T') month = 10;
                if (a == 'N' && b == 'O' && c == 'V') month = 11;
                if (a == 'D' && b == 'E' && c == 'C') month = 12;
                if (month == 0) error (1, -1, "Unrecognized month\n");
                /* printf ("Day %d, month %d year %d\n", day, month, year); */
                dates[n] = day ^ (month << 5) ^ (year << 9);
                n++;
            } while (FOREVER);
        }
    void names_to_hash (HS_struct *hs)
    {
        int i;
        for (i = 0; i < N_seq; i++)
            HS_insert (hs, &names[i*MAX_NAME_LEN]);
    }
    void read_pairs (UF_struct *uf, HS_struct *hs, FILE *fp)
    {
        char st[500], name1[50], name2[50], sign;
        int i = 0, pos1, pos2;
        while (fgets (st, MAX_LINE_SIZE, fp) != NULL) {
            sscanf (st, "%s %s %c", name1, name2, &sign);
            if ((pos1 = HS_find (hs, name1)) == -1)
                error (1, -1, "Can't find %s in hash table\n", name1);
            if ((pos2 = HS_find (hs, name2)) == -1)
                error (1, -1, "Can't find %s in hash table\n", name2);
            if (UF_find (uf, pos1) != UF_find (uf, pos2))

```

```

UF_union (uf, pos1, pos2);
/* else printf ("%s and %s are already clustered\n", name1, name2); */
printf ("Read %d pairs\n", i);
}
/*=====
void print_sizes (UF_struct *uf)
{
    int i;
    for (i = 0; i < N_seq; i++)
        if (uf->fathers[i] == i)
            printf ("%d\n", uf->size[i]);
}
/*=====
void print_clusters (UF_struct *uf, HS_struct *hs)
{
    int i, j, k, l, year, month, day;
    char str1[MAX_NAME_LEN*2], str2[MAX_NAME_LEN*2], str3[MAX_NAME_LEN*2];
    for (i = 0; i < N_seq; i++) {
        /* Get the name of the sequence */
        for (j = 0; names[MAX_NAME_LEN*i+j] != '\0' && j < MAX_NAME_LEN; j++)
            str1[j] = names[MAX_NAME_LEN*i+j];
            str1[j] = '\0';
        /* Get the name of the cluster */
        k = cl_names[UF_find (uf, i)];
        for (j = 0; names[MAX_NAME_LEN*k+j] != '\0' && j < MAX_NAME_LEN; j++)
            str2[j] = names[MAX_NAME_LEN*k+j];
            str2[j] = '\0';
        /* Get the date of the sequence */
        l = dates[i];
        year = (l >> 9) + 1950;
        month = (l >> 5) & 0xf;
        day = (l & 0x1f);
        fprintf (ofp, "%s %s %02d-%02d-%04d %7d\n",
            str1, str2, day, month, year, uf->size[UF_find (uf, i)]);
    }
}
/*=====
void set_names (UF_struct *uf, HS_struct *hs)
{
    int i, j;
    /* Go over the sequences */
    for (i = 0; i < N_seq; i++) {
        /* Look at the root of the tree */
        j = UF_find (uf, i);
        if (cl_names[j] == -1 || /* Not set yet */
            dates[cl_names[j]] > dates[i] || /* our date is older */
            /* they are of the same date but ours precedes lexicographically */
            (dates[cl_names[j]] == dates[i] &&
            strcmp (&names[cl_names[j]*MAX_NAME_LEN], &names[i*MAX_NAME_LEN],
            MAX_NAME_LEN) > 0))

```

```
cl_names[j] = i;  
}
```



```

#include <stdio.h>
#include <math.h>
#include "union_find.h"

unsigned int UF_find(UF_struct *uf, unsigned int ind)
{
    int ind1 = ind, tmp;

    /* Find root */
    while ( uf->fathers[ind1] != ind )
        ind = uf->fathers[ind1];

    /* path compression */
    while ( uf->fathers[ind1] != ind ) {
        tmp = ind1;
        ind1 = uf->fathers[ind1];
        uf->fathers[tmp] = ind;
    }

    return ind;
}

int UF_size(UF_struct *uf, unsigned int ind)
{
    unsigned int tmp;

    tmp = UF_find( uf, ind );
    return(uf->size[tmp]);
}

void UF_union(UF_struct *uf, unsigned int ind1, unsigned int ind2)
{
    unsigned int tmp, tmp1, tmp2;

    /* Get the roots */
    tmp1 = UF_find( uf, ind1 );
    tmp2 = UF_find( uf, ind2 );

    /* Find smallest depth */
    if ( uf->depth[tmp1] < uf->depth[tmp2] ) {
        tmp = tmp1;
        tmp1 = tmp2;
        tmp2 = tmp;
    }

    /* Fix */
    uf->fathers[tmp2] = tmp1;
    if ( uf->depth[tmp1] == uf->depth[tmp2] )
        uf->depth[tmp1]++;
    uf->size[tmp1] += uf->size[tmp2];
}

void UF_init( int size, UF_struct *uf )
{
    unsigned int i;

    *uf = (UF_struct *) malloc( sizeof( UF_struct ) );
    if ( *uf == NULL )
        error(1,0, "Can't allocate union-find structure");
}

```

```

(*uf)->fathers = (unsigned int *)malloc(sizeof(unsigned int)*size);
if ( (*uf)->fathers == NULL )
    error(1,0, "Can't allocate fathers");
for ( i = 0; i < (unsigned int)size; ++i )
    (*uf)->fathers[i] = i;

(*uf)->depth = (unsigned int *)malloc(sizeof(unsigned int)*size);
if ( (*uf)->depth == NULL )
    error(1,0, "Can't allocate depth");
memset((*)uf->depth, 0, size * sizeof(unsigned int));

(*uf)->size = (int *)malloc(sizeof(int)*size);
if ( (*uf)->size == NULL )
    error(1,0, "Can't allocate sizes array of union find");
for ( i = 0; i < size; i++)
    (*uf)->size[i] = 1;
}

void UF_destroy (UF_struct *uf)
{
    if ( uf == NULL )
        return;
    free( uf->depth );
    free( uf->size );
    free( uf->fathers );
    free( uf->size );
    free( uf );
}

```


Sun Aug 9 10:41:15 1998

Listing for Adam Sartiel

```
/*
 * Header file for hash package
 */

/* Macros */

#ifndef NULL
#define NULL 0
#endif

typedef struct el_char {
    char *key;
    unsigned data;
    struct el_char *next;
} el_char;

typedef struct HS_struct {
    int real_size;
    int size;
    el_char **hash;
} HS_struct;

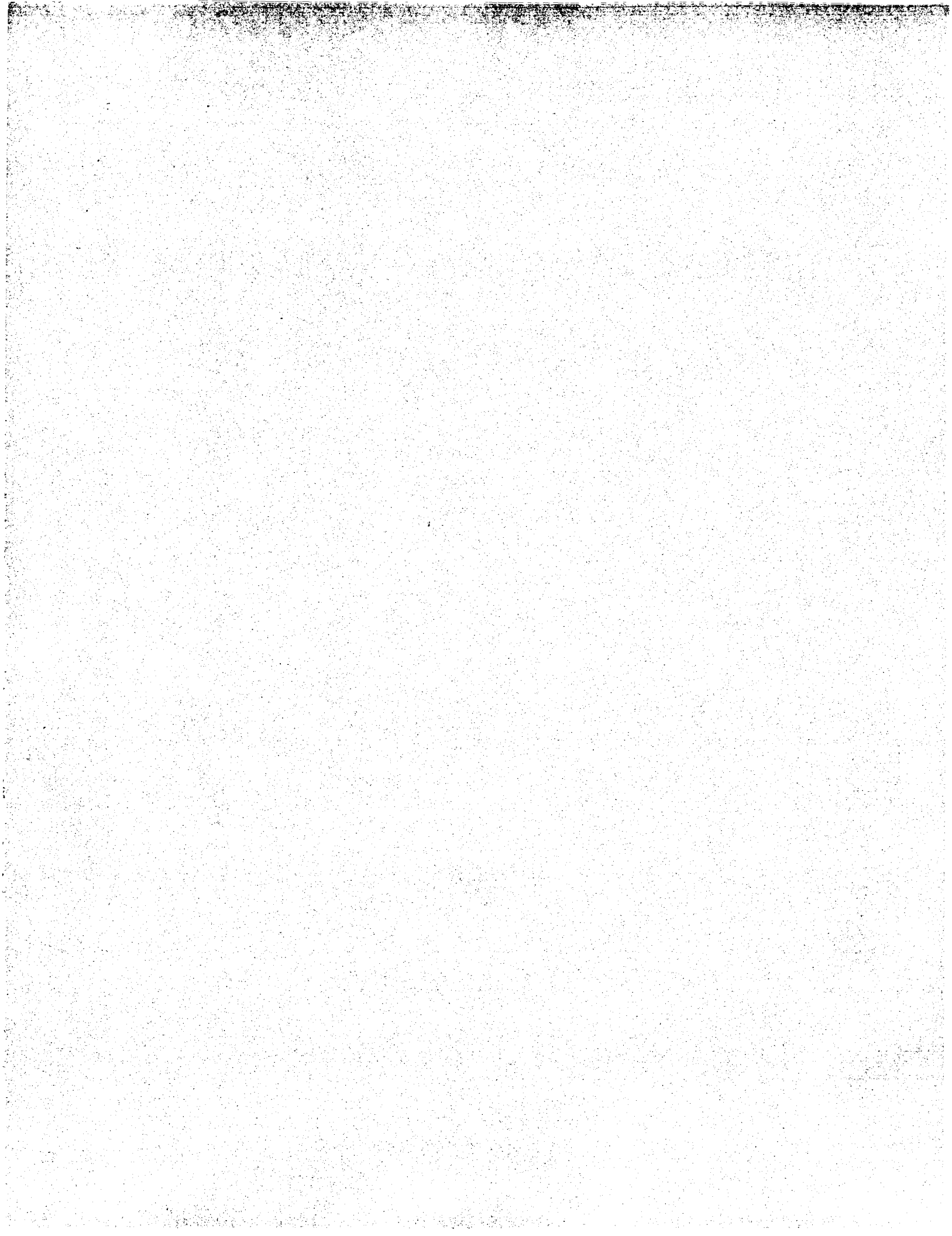
/* Function prototypes */

HS_struct *init_hash (int size);
void HS_destroy (HS_struct *hs);
void HS_insert (HS_struct *hs, const char *key);
int HS_find (HS_struct *hs, const char *key);
int HS_find_or_insert (HS_struct *hs, const char *key);
void HASH_print_table (HS_struct *hs);
el_char *hash_st_next_element (HS_struct *hs, el_char *elem, int *i);
```

hash.h


```
typedef struct UF_struct {
    unsigned int *depth;
    unsigned int *fathers;
    int *size;
} UF_struct;

int UF_size (UF_struct *uf, unsigned int ind);
unsigned int UF_find(UF_struct *uf, unsigned int ind);
void UF_union(UF_struct *uf, unsigned int ind1, unsigned int ind2);
void UF_init( int size, UF_struct **uf );
void UF_destroy (UF_struct *uf);
```



Using for Adam Sartiell

Sun Aug 9 10:41:16 1998

```

CC = cc
OBJ = hash_st_${ARCH}.o union_find_${ARCH}.o
SRC = hash_st.c union_find.c
INC = hash_st.h union_find.h
CFLAGS = -c -O -I$(PRODDHOME)/include
CL = cc
LFLAGS = -L$(PRODDHOME)/lib/${ARCH}obj -lm
LPRM = -lprn

all: make_names.${ARCH}

make_names.${ARCH}: make_names_${ARCH}.o $(OBJ) $(SRC) $(INC) Makefile
$(CL) make_names_${ARCH}.o $(OBJ) $(LFLAGS) $(LPRM) -o make_names_${ARCH}

make_names_${ARCH}.o: make_names.c $(INC) Makefile
$(CC) $(CFLAGS) make_names.c -o make_names_${ARCH}.o
hash_st_${ARCH}.o: hash_st.c hash_st.h Makefile
$(CC) $(CFLAGS) hash_st.c -o hash_st_${ARCH}.o
union_find_${ARCH}.o: union_find.c union_find.h Makefile
$(CC) $(CFLAGS) union_find.c -o union_find_${ARCH}.o

depend: $(SRCS)
makedepend $(CFLAGS) $(SRCS) $(OBJS)

clean:
rm -f *${ARCH}.o make_names.${ARCH} core

install:
cp make_names.${ARCH} ${PRODDHOME}/bin/${ARCH}/make_names

# DO NOT DELETE THIS LINE -- make depend depends on it.

```

Makefile

Sun Aug 9 10:42:56 1993

Listing for Adam Sartiell

```
#include <stdio.h>
#include <varargs.h>

/* this implementation ignores err, that should print the errno string */

void error(ex, err, format, va_alist)
int ex;
int err;
char *format;
va_dcl
{
    va_list ap;
    va_start(ap);
    vfprintf(stderr, format, ap);
    if (ex != 0)
        exit(ex);
    va_end(ap);
}

void sdbg() {}
```



```
static char *rcsid = "$Header: /tmp_mnt/home/users/gidi/util/lib/prm/RCS/match.c
,v 1.4 92/07/23 16:55:31 gidi Exp Locker: gidi $";
```

```
/*
 * $Log: match.c,v $
 * Revision 1.4 92/07/23 16:55:31 gidi
 * *** empty log message ***
 *
 * Revision 1.3 92/01/27 13:28:11 gidi
 * uflag and field are joined
 *
 * Revision 1.2 91/11/15 14:57:31 gidi
 * match strings package
 */
```

```
#include <ctype.h>
#define NULL 0
static char **match_result_string = NULL;
```

```
void match_init(s)
char **s;
{
    match_result_string = s;
}
```

```
static char *strndup(s, n)
char *s;
int n;
```

```
{
    char *b;
    b = (char *)realloc(NULL, n+1);
    bcopy(s, b, n);
    b[n] = '\0';
    return b;
}
```

```
char *match_number(s)
char *s;
```

```
{
    char *b = s;
    if (!isdigit(*s))
        return NULL;
    while(isdigit(*s))
        s++;
}
```

```
if (match_result_string != NULL)
    match_result_string = strndup(b, s-b);
return s;
}
```

```
char *match_alpha(s)
char *s;
```

```
{
```

```
char *b = s;
if (!isalpha(*s))
    return NULL;
while(isalnum(*s))
    s++;
if (match_result_string != NULL)
    match_result_string = strndup(b, s-b);
```

```
return s;
}
```

```
char *match_name(s)
char *s;
```

```
char *b = s;
if (!isalnum(*s) && *s != '_' && *s != '-' && *s != '-')
    return NULL;
while(isalnum(*s) || *s == '_' || *s == '-')
    s++;
```

```
if (match_result_string != NULL)
    match_result_string = strndup(b, s-b);
return s;
}
```

```
char *match_space1(s)
char *s;
```

```
char *b = s;
if (!isspace(*s))
    return NULL;
while(isspace(*s))
    s++;
```

```
if (match_result_string != NULL)
    match_result_string = NULL;
return s;
}
```

```
char *match_space(s)
char *s;
```

```
{
    char *b = s;
    if (!isspace(*s))
        return s;
    while(isspace(*s))
        s++;
    if (match_result_string != NULL)
```

match.c

match.c

A-323

```

    *match_result_string = NULL;
    return s;
}

char *match_nspace(s)
char *s;
{
    char *b = s;
    if (isspace(*s))
        return NULL;
    while(!isspace(*s))
        s++;
    if (match_result_string != NULL)
        *match_result_string = strdup(b, s-b);
    return s;
}

char *match_string(s)
char *s;
{
    char *b = s;
    if (*s == '\0')
        return 0;
    if (*s == '"') {
        s++;
        while(*s != '"' && *s != '\0')
            s++;
        if (*s != '"')
            return NULL;
        if (match_result_string != NULL)
            *match_result_string = strdup(b+1, s-b-1);
        return s+1;
    }
    if (isspace(*s))
        return NULL;
    s++;
    while(isspace(*s) && !isspace(*s))
        s++;
    if (match_result_string != NULL)
        *match_result_string = strdup(b, s-b);
    return s;
}

char *match_char(s)
char *s;
{
    char *b = s;

```

```

    if (!isgraph(*s))
        return NULL;
    if (match_result_string != NULL)
        *match_result_string = strdup(b, 1);
    return s+1;
}

char *match_format(s)
char *s;
{
    char *b = s;
    printf("%s\n", s);
    if (*s != '%')
        return NULL;
    s++;
    if (*s != '[') {
        printf("name\n");
        if ((s = match_name(s)) == NULL)
            return NULL;
    }
    if (match_result_string != NULL)
        *match_result_string = strdup(b, s-b);
    return s;
}

```

```

static char *rcsid = "$Header: /tmp_rmt/home/users/gidi/util/lib/prm/RCS/prm.c.v
1.4 92/07/23 16:55:17 gidi Exp Locker: gidi $";

/*
 * $Log: prm.c,v $
 * Revision 1.4 92/07/23 16:55:17 gidi
 * *** empty log message ***
 *
 * Revision 1.3 92/01/27 13:26:59 gidi
 * *** empty log message ***
 *
 * Revision 1.2 91/11/15 14:54:50 gidi
 * new prm_argv
 *
 * Revision 1.1 91/11/15 14:52:57 gidi
 * Initial revision
 *
 */

/*****/
/* */
/* prm */
/* */
/*****/

#include <varargs.h>
#include <fcntl.h>
#include <unistd.h>
#include <ctype.h>
#include <string.h>
#include "prm.h"

static prm_inst *prms(PR_MAX_PRM);
static arg_inst *args;
static arg_inst *prev_arg;

static prm_inst *prm_construct();
static arg_inst *arg_construct();
static void prm_destruct();
static void arg_destruct();
static void print_result();
static void print_help();
static void print_error();
void parse_file();
void parse_buf();

char *prm_string_dup();
static char *program_name;
static int help_flag;
static int arg_count;

/* added by Raveh, 5.1.97 */
static int parse_prm(va_list ap);
static int parse_arg(int argc, char *argv());
static int extract_data(int flags);

/**** prm_argv ****/

```

```

int prm_argv (argc, argv, flags, va_alist)
int argc;
char *argv[];
int flags;
va_dcl
{
    va_list ap;
    va_start(ap);

    sdbg("f", 0, "prm_argv");

    match_init(&MatchStr);
    program_name = argv[0];
    prev_arg = NULL;
    arg_count = 0;
    help_flag = 0;
    args = NULL;

    if (argc > 1)
        if (strcmp(argv[1], "!") == 0)
            help_flag = 1;

    if (!parse_prm(ap))
        error (1, 0, "parameter error");

    if (!parse_arg(argc-1, &argv[1]))
        error (1, 0, "argument error");

    if (!extract_data (flags))
        error (1, 0, "can't extract data");

    va_end(ap);
}

/**** prm_buf ****/

int prm_buf (buf, len, flags, va_alist)
char *buf;
int len;
int flags;
va_dcl
{
    int argc;
    char **argv;
    va_list ap;
    va_start(ap);

    sdbg("f", 0, "prm_buf");

    program_name = "buffer";
    prev_arg = NULL;
    arg_count = 0;
    help_flag = 0;

    if (!parse_prm(ap))
        error (1, 0, "parameter error");

    parse_buf (buf, len, &argc, &argv);
}

```

```

    if (!parse_arg(argc, argv))
        error(1, 0, "argument error");

    if (!extract_data(flags))
        error(1, 0, "can't extract data");

    va_end(ap);
}

/***** parse_prm ****/
static int parse_prm(ap)
    va_list ap;
{
    char *def;
    int i;
    prm_inst *cur;
    prm_inst *prev[PR_MAX_PRM];
    char *ndef;
    int c;
    prm_inst *p;

    sdbg("f", 0, "parse_prm");

    /* pad parameters linked list with NULL */
    for (i = 0; i < PR_MAX_PRM; i++) {
        prms[i] = NULL;
        prev[i] = NULL;
    }

    /* Main Loop:
     * 1. construct a parameter instance
     * 2. identify parameter type
     * 3. parse parameter according to the type
     * 4. add parameter to the linked list
     * 5. goto (1)
     */
    while ((def = va_arg(ap, char *)) != EOLIST) {
        /* allocate a parameter instance */
        if ((cur = prm_construct()) == NULL)
            return 0;

        /* parse the parameter common data */
        if (!prm_parse_inst(def, &ndef, &ap, cur))
            return 0;

        /* identify parameter type */
        c = 0;
        for (i = 0; i < PR_MAX_PRM; i++)
            if (!prms[i].ident(ndef))
                c++;
    }
}

```

prm.c

```

    cur->type = i;
    break;
}

if (c != 1)
    error(0, -1, "can't identify type of '%s'", def);
return 0;
}

/* parse parameter */
if (!(*prms[cur->type].parse)(ndef, &ap, cur))
    return 0;

sdbg("p", 0, "%s: %s %x %d '%s'",
    prms[cur->type].name,
    ndef,
    cur->addr,
    cur->nelems,
    cur->comment);

/* add current parameter (cur) to the parameters link list */
if (prms[cur->type] == NULL) {
    prms[cur->type] = cur;
    cur->prev = NULL;
}
else {
    prev[cur->type]->next = cur;
    cur->prev = prev[cur->type];
}

prev[cur->type] = cur;

return 1;
}

/***** parse_arg ****/
static int parse_arg(argc, argv)
    int argc;
    char *argv[];
{
    int i, j;
    int c;
    int type;
    arg_inst *cur;

    sdbg("f", 0, "parse_arg");

    /* Main Loop:
     * 1. if the argument is <file-name> then get arguments from the file
     * 2. construct an argument instance
     * 3. parse argument according to the type
     * 4. add argument to the linked list
     * 5. goto (1)
     */
}

```

prm.c

A-326

```

for ( i = 0 ; i < argc ; i++ ) {
    /* get arguments from a file */
    if (argv[i][0] == '@') {
        parse_file(&argv[i][1]);
        continue;
    }
    /* construct an argument instance */
    if ((cur = arg_construct(argv[i])) == NULL)
        return 0;
    /* add argument to the linked list */
    if (args == NULL) {
        args = cur;
        cur->prev = NULL;
    }
    else
        prev_arg->next = cur;
    prev_arg = cur;
    return 1;
}

/**** extract_data ****/
static int extract_data(flags)
    int flags;
{
    int i;
    prm_inst *p, *p_tmp;
    arg_inst *a, *a_tmp;

    sdbg("f", 0, "extract_data");

    /* if the flag P_HELP appears and help_flag is set then print help and exit */
    if ((flags & P_HELP) && help_flag) {
        print_help();
        exit(0);
    }

    /* get values from arguments
    * start with parameters of type 0 upto PR_MAX_PRM-1
    *
    * The source of the values is in the following order:
    * 1. argv (or parameters files)
    * 2. external function
    * 3. parameter default
    */
    for ( i = 0 ; i < PR_MAX_PRM ; i++ )

```

prm.c

```

for (p = prms[i] ; p != NULL ; p = p->next) {
    /* for each unused argument try to get data */
    for (a = args ; a != NULL ; a = a->next) {
        if (a->used)
            continue;
        /* check whether the parameter is still free */
        if (!(*prm_types[i].get)(p, D_FREE, NULL))
            break;
        (void) (*prm_types[i].get)(p, D_ARG, &a);
    }
    /* if any data was retrieved then next parameter */
    if (p->nelems > 0)
        continue;
    /* try to get data from external source */
    if (p->external)
        if (!(*prm_types[i].get)(p, D_EXTERN, NULL))
            continue;
    /* try to get data from the parameter's default */
    if (!(*flags & P_IGNORE))
        if (!(*prm_types[i].get)(p, D_DEFAULT, NULL))
            continue;
}

/* print error, if necessary */
if (flags & P_ERROR)
    print_error();

/* print the parameters final values, if necessary */
if (flags & P_PRINT)
    print_result();

/* free parameters linked list */
for ( i = 0 ; i < PR_MAX_PRM ; i++ )
    for ( p = prms[i] ; p != NULL ; ) {
        p_tmp = p;
        p = p->next;
        prm_destruct(p_tmp);
    }

/* free arguments list */
for (a = args ; a != NULL ; ) {
    a_tmp = a;
    a = a->next;
    arg_destruct(a_tmp);
}

```

prm.c

A-327

```

    return 1;
}

/*****
**** prm_parse_inst ****/
int prm_parse_inst(def, ndef, ap, cur)
char *def;
char **ndef;
val_list *ap;
prm_inst *cur;
{
    char *comment;
    char *s;
    int quote = 0;

    /* the check of comment should be changed because the ! sign can appear before
    the comment */

    cur->comment = NULL;
    comment = def;
    while (*comment != '\0')
    {
        switch (*comment) {
            case ' ':
                quote ^= 1;
                break;
            case '!':
                if (quote == 0) {
                    cur->comment = _prm_string_dup(comment+1);
                    *comment = '\0';
                }
                break;
            case '\n':
                comment++;
                if (cur->comment != NULL)
                    break;
        }

        s = (char *)match_space(def);
        if (*s == '^') {
            cur->external = 1;
            s++;
            s = (char *)match_space(s);
        }
        else
            cur->external = 0;

        if ((*ndef = _prm_string_dup(s)) == NULL)
            error(1, -1, "prm_string_dup error");
        if ((cur->addr = va_arg(*ap, void *)) == NULL)
            return 0;

        return 1;
    }

    /**** parse_file ****/

```

```

void parse_file(name)
char *name;
{
    char *buf;
    int fd;
    int len;
    int argc;
    char **argv;
    int i;

    sdbg("f", 0, "parse_file");
    if ((fd = open(name, O_RDONLY)) < 0)
        error(1, -1, "can't open parameters file: %s", name);
    if ((len = lseek(fd, 0, SEEK_END)) == -1)
        error(1, -1, "can't seek to end of parameters file: %s", name);
    buf = (char *) realloc(NULL, len * sizeof(char));

    if (lseek(fd, 0, SEEK_SET) == -1)
        error(1, -1, "can't seek to start of parameters file: %s",
            name);
    if (read(fd, buf, len) < len)
        error(1, -1, "can't read parameters file: %s", name);
    parse_buf(buf, len, &argc, &argv);
    parse_arg(argc, argv);
    close(fd);
}

/**** parse_buf ****/
void parse_buf(buf, len, argc, argv)
char *buf;
int len;
int *argc;
char ***argv;
{
    register int i, j;
    register int c = 0;
    int in_quote = 0;
    int in_comment = 0;
    char *last;

    sdbg("f", 0, "parse_buf");
    for (i = 0; i < len; i++)
        if (buf[i] == ',')
            c++;
    *argv = (char **) realloc(NULL, c * sizeof(char *));
    *argc = 0;
    last = NULL;

```

```

for ( i = 0, j = 0; i < len; i++) {
    if (in_comment && buf[i] != '\n')
        continue;
    switch(buf[i]) {
        case '\t':
            case '\n':
            case '\r':
                if (in_comment && buf[i] == '\n')
                    in_comment = 0;
                if (in_quote) {
                    if (last == NULL)
                        last = &buf[j];
                    buf[j++] = buf[i];
                }
                break;
            case '#':
                if (i > 0) {
                    if (buf[i-1] == '\n')
                        in_comment = 1;
                    else {
                        if (last == NULL)
                            last = &buf[j];
                        buf[j++] = buf[i];
                    }
                }
                else
                    in_comment = 1;
                break;
            case '.':
                in_quote ^= 1;
                break;
            case ',':
                buf[j++] = '\0';
                if (last != NULL)
                    if (last != &buf[j])
                        (*argc)++;
                last = NULL;
                break;
            default:
                if (last == NULL)
                    last = &buf[j];
                buf[j++] = buf[i];
                break;
        }
    }
}

/**** prm_destruct ****/
static void prm_destruct(p)
    prm_inst *p;
{
    if (p->prev == NULL)
        prms[p->type] = p->next;
    else
        p->prev->next = p->next;
    if (p->next != NULL)

```

```

        p->next->prev = p->prev;
        (*prm_types[p->type].destruct)(p);
        free(p);
    }
    /**** arg_destruct ****/
    static void arg_destruct(a)
        arg_inst *a;
    {
        if (a->prev == NULL)
            args = a->next;
        else
            a->prev->next = a->next;
        if (a->next != NULL)
            a->next->prev = a->prev;
        free(a);
    }
    /**** prm_construct ****/
    static prm_inst *prm_construct()
    {
        prm_inst *p;
        p = (prm_inst *)realloc(NULL, sizeof(prm_inst));
        p->nelems = 0;
        p->next = NULL;
        return p;
    }
    /**** arg_construct ****/
    static arg_inst *arg_construct(value)
        char *value;
    {
        arg_inst *a;
        a = (arg_inst *)realloc(NULL, sizeof(arg_inst));
        a->used = 0;
        a->count = arg_count++;
        if ((a->value = _prm_string_dup(value)) == NULL)
            error(1, -1, "prm_string_dup error");
        return a;
    }
    /**** print_result ****/
    static void print_result()
    {
        int i;
        prm_inst *p;

```



```

    sdbg("f", 0, "print_result");
    for ( i = 0 ; i < PR_MAX_PRM ; i++ )
        for ( p = prms[i] ; p != NULL ; p = p->next )
            (*prm_types[i].print)(p);
}

/**** print_error ****/
static void print_error()
{
    int i;
    arg_inst *a;

    sdbg("f", 0, "print_error");

    fprintf (PrmFp, "Unused arguments\n");
    fprintf (PrmFp, "-----\n");
    for ( a = args ; a != NULL ; a = a->next )
        if ((a->used)
            fprintf (PrmFp, "%s\n", a->value);
        exit(1);
    }

/**** print_help ****/
static void print_help()
{
    int i;
    prm_inst *p;

    sdbg("f", 0, "print_help");

    fprintf (PrmFp, "\nusage : %s arg1 arg2 ... \n\n", program_name) ;
    fprintf (PrmFp, "arguments available : \n\n" );
    for ( i = 0 ; i < PR_MAX_PRM ; i++ )
        for ( p = prms[i] ; p != NULL ; p = p->next )
            (*prm_types[i].help)(p);

    fprintf (PrmFp, "\n* the arguments can be used in any order .\n" );
    fprintf (PrmFp, "*** [...] means the default value .\n\n" );
}

```

```
static char *rcsid = "$Header: /tmp_mnt/home/users/gidi/util/lib/prm/RCS/prm_fun
cs.c,v 1.5 92/07/23 16:55:25 gidi Exp Locker: gidi $";
```

```
/*
 * $Log: prm_funcs.c,v $
 * Revision 1.5 92/07/23 16:55:25 gidi
 * *** empty log message ***
 * Revision 1.4 92/01/27 13:28:00 gidi
 * uflag and field are joined
 * Revision 1.3 91/11/18 16:39:07 gidi
 * debugging
 * Revision 1.2 91/11/15 14:57:01 gidi
 * prm_argv additional functions
 */
```

```
#include <varargs.h>
#include <ctype.h>
#include <string.h>
#define prm_c
#include "prm.h"

extern char *match_number();
extern char *match_alpha();
extern char *match_format();
extern char *match_space();
extern char *match_spacer();
extern char *match_nspace();
extern char *match_string();
extern char *match_char();
extern char *match_name();

char *index();
char *_prm_string_dup();

/* Added by Raveh, 5.1.97 */
static int prepare_argv(char **argv, char *s, char *delimit);
static int strtokcount(char *s, char *delimit);
static int elsetos(char *format, void *s, char *t);
static equal(char *s1, char *s2);
static equaln(char *s1, char *s2, int n);
static data_size(char *format);
static prm_sscanf(char *v, char *f, void *d);
static print_help_line(char *def, char *fmt, char *comment);
```

```
/******
 * index */
/******

static int vp_index_number;
static char *vp_index_format;

int p_index_ident(b)
char *b;
{
```

```
char *s;
if (b[0] != '#')
return 0;

if ((s = match_number(b+1)) == NULL)
return 0;
vp_index_number = atoi(MatchStr);
if ((s = match_spacer(s)) == NULL)
return 0;
if ((s = match_format(s)) == NULL)
return 0;
vp_index_format = MatchStr;
s = match_space(s);
if (*s != '\0')
return 0;
return 1;
}

int p_index_print(p)
prm_inst *p;
{
char target[256];

elsetos(p->p.index.format, p->addr, target);
fprintf(prmfp, "%s argument %d\n", p->p.index.arg_index);
fprintf(prmfp, "%s\n", target);
}

int p_index_parse(b, ap, p)
char *b;
va_list *ap;
prm_inst *p;
{
p->p.index.arg_index = vp_index_number;
p->p.index.format = vp_index_format;
return 1;
}

int p_index_get(p, flag, a)
prm_inst *p;
int flag;
arg_inst **a;
char s[256];
{
switch(flag) {
case D_FREE:
if (p->nelems == 0)
return 1;
return 0;

case D_ARG:
if ((*a)-->count != p->p.index.arg_index-1)
return 0;
prm_sscanf((*a)-->value, p->p.index.format, p->addr);
(*a)-->used = 1;
p->nelems++;
return 1;
}
```

```

        case D_EXTERN:
            return 0;
        case D_DEFAULT:
            return 0;
        default:
            return 0;
    }

    int p_index_help(p)
    prm_inst *p;
    {
        char str[1024];

        sprintf(str, "%d", p->p.index.arg_index);

        print_help_line(str, p->p.index.format, p->p.comment);
    }

    int p_index_destruct()
    {
        /******
        */
        /* field */
        /* *****/
        /******

        static char *vp_field_name;
        static char *vp_field_default;
        static char *vp_field_format;
        static int vp_field_array;

        /* p_field_ident - match a field definition with :
        * <name><space><space>[<string><space>]<format><space>[*]
        * and set :
        * a. vp_field_name to <name>
        * b. vp_field_default to <string>
        * c. vp_field_format to <format>
        * d. vp_field_array to 1 if '*', appears and 0 otherwise
        * return :
        * 0 if mismatched
        * 1 if matched
        */

        int p_field_ident(b)
        char *b;
        {
            char *s;

            /* match field name */

            if ((s = match_name(b)) == NULL)
                return 0;
            vp_field_name = MatchStr;

            /* match equal sign */

            s = match_space(s);

```

```

        if (*s != '=')
            return 0;
        s = match_space(s+1);

        /* match default string */

        /* printf("before def\n"); */

        vp_field_default = NULL;

        if (*s != '%')
            if ((s = match_string(s)) != NULL)
                vp_field_default = MatchStr;
            if ((s = match_space(s)) == NULL)
                return 0;
        }

        /* printf("after def\n"); */

        /* match format clause */

        if ((s = match_format(s)) == NULL)
            return 0;
        vp_field_format = MatchStr;

        s = match_space(s);

        /* printf("before array\n"); */

        /* match array sign */

        vp_field_array = 0;

        if (*s == '*') {
            s = match_space(s+1);
            vp_field_array = 1;
        }
        else if (*s != '\0')
            return 0;

        return 1;
    }

    /* p_field_print - print field in the format :
    * <field-name> = <value-1> [, <value-2> ...] ;
    */

    int p_field_print(p)
    prm_inst *p;
    {
        char target[256];
        void *address;
        int i;

        fprintf(PrmFp, "%s = ", p->p.field.name);
        for (i = 0; i < p->p.nelems; i++) {
            address = (void *) ((char *) p->addr +
                                (p->p.field.elements[i]));
            elsetos(p->p.field.format, address, target);

```

```

    if ( i > 0 )
        putc(' ', prmfp);
    fprintf(prmfp, " %s", target);
}
fprintf(prmfp, "\n");
}

/* p_field_parse - initialize field structure */
int p_field_parse(b, ap, p)
char *b;
va_list *ap;
prm_inst *p;
{
    p->p.field.name = vp_field_name;
    p->p.field.def = vp_field_default;
    p->p.field.format = vp_field_format;
    p->p.field.elements = data_size(p->p.field.format);
    /* set array size to 1, if the array sign (('') wasn't given */
    if (vp_field_array)
    {
        p->p.field.n_addr = va_arg(*ap, int *);
        p->p.field.maxelems = *(p->p.field.n_addr);
    }
    else
    {
        p->p.field.n_addr = NULL;
        p->p.field.maxelems = 1;
    }
    return 1;
}

/* p_field_get - get data into the field structure */
int p_field_get(p, flag, a)
prm_inst *p;
int flag;
arg_inst **a;
{
    void *address;
    int argc;
    char **argv;
    int i;
    char *extern_data;
    int follow_arg;
    char *in;
    switch(flag)
    {
        /* check whether there are unused elements in the array */
        case D_FREE:
            if (p->p.nelems < p->p.field.maxelems)

```

```

    return 1;
return 0;
/* get data from the arguments list */
case D_ARG:
    /* check whether the field name matches the argument */
    in = index((*a)->value, '=');
    if ((in = index((*a)->value, '=')) != NULL)
    {
        if (!equal((*a)->value, p->p.field.name,
            in-(*a)->value))
            return 0;
        if ((*in+1) == '\0')
            follow_arg = TRUE;
        else
            follow_arg = FALSE;
    }
    else
    {
        if ((*a)->value[0] != '-')
            return 0;
        if (!equal((*a)->value+1, p->p.field.name))
            return 0;
        follow_arg = TRUE;
    }
    (*a)->used = TRUE;
    /* if a is of the form "<field>" then use the following
       arguments as data */
    if (follow_arg)
    {
        /* get data, until the array is full or until the
           end of the arguments list */
        while((p->p.nelems < p->p.field.maxelems) &&
            ((*a)->next != NULL))
        {
            /* construct the argument string */
            (*a) = (*a)->next;
            if ((*a)->used)
                break;
            (*a)->used = TRUE;
            /* if the argument is a mark of end of list
               then break the loop */
            if (strcmp((*a)->value, ";") == 0)
                break;
            /* get data into the address */
            address = (void *) ((char *)p->addr+
                (p->p.field.elements*p->p.nelems));
            prm_sscanf((*a)->value, p->p.field.format,
                address);
            p->p.nelems++;
        }
    }
}

```

```

    }

    /* else, get data from the value field of the argument */
    else {
        argc = prepare_argv(&argv, in+1, "");

        for (i = 0; i < argc && p->nelems < p->p.field.maxelems; i++) {
            address = (void *) ((char *) p->addr +
                (p->p.field.elsize * p->nelems));
            prm_sscanf(argv[i], p->p.field.format, address);
            p->nelems++;
        }

        (*a)->used = TRUE;
        return 1;
    }

    /* get data from external source */
    case D_EXTERN:
        if (PrmExternFunc == NULL)
            return 0;

        /* get external data string */
        if ((extern_data = (*PrmExternFunc)(PR_FIELD, &p->p.field)) == NULL)
            return 0;

        /* prepare arguments vector from extern_data */
        argc = prepare_argv(&argv, extern_data, "");

        /* for each argument, calculate its address, and get data */
        for (i = 0; i < argc && p->nelems < p->p.field.maxelems; i++) {
            address = (void *) ((char *) p->addr +
                (p->p.field.elsize * p->nelems));
            prm_sscanf(argv[i], p->p.field.format, address);
            p->nelems++;
        }
        return 1;
    }

    /* get data from the default string */
    case D_DEFAULT:
        /* prepare arguments vector from the default string */
        argc = prepare_argv(&argv, p->p.field.def, "");

        /* for each argument, calculate its address, and get data */
        for (i = 0; i < argc && p->nelems < p->p.field.maxelems; i++) {
            address = (void *) ((char *) p->addr +
                (p->p.field.elsize * p->nelems));
            prm_sscanf(argv[i], p->p.field.format, address);
            p->nelems++;
        }

```

```

    }

    return 1;
default:
    return 0;
}

int p_field_help(p)
    prm_inst *p;
{
    char target[256];
    int argc;
    char **argv;
    int i;
    void *address;
    char str[1024];

    sprintf(str, "%s = [", p->p.field.name);
    if (p->p.field.def != NULL) {
        argc = prepare_argv(&argv, p->p.field.def, "");
        for (i = 0; i < argc; i++) {
            if (i != 0)
                strcat(str, " ");
            address = (void *) p->addr;
            prm_sscanf(argv[i], p->p.field.format, address);
            elseos(p->p.field.format, address, target);
            strcat(str, target);
        }
        strcat(str, "];");
    }
    print_help_line(str, p->p.field.format, p->comment);

    int p_field_destruct(p)
        prm_inst *p;
    {
        if (p->p.field.n_addr != NULL)
            *p->p.field.n_addr = p->nelems;
    }

    /******
     * flag */
    /******
    static char *vp_flag_name;
    static int vp_flag_def;

    int p_flag_ident(p)
        char *b;
    {
        char *s;
        if ((s = match_name(b)) == NULL)
            return 0;
        vp_flag_name = MatchStr;
    }

```

```

s = match_space(s);
if (*s == '-')
    vp_flag_def = 0;
else if (*s == '\0')
    vp_flag_def = 1;
else
    return 0;
return 1;
}

int p_flag_print(p)
    prm_inst *p;
{
    if (*(int *)p->addr)
        fprintf(prmFp, "%s\n", p->p.flag.name);
}

int p_flag_parse(b, ap, p)
    char *b;
    va_list *ap;
    prm_inst *p;
{
    p->p.flag.name = vp_flag_name;
    p->p.flag_def = vp_flag_def;
    return 1;
}

int p_flag_get(p, flag, a)
    prm_inst *p;
    int flag;
    arg_inst **a;
{
    char *extern_data;
    switch(flag)
    case D_FREE:
        if (p->nelems == 0)
            return 1;
        return 0;
    case D_ARG:
        if (!equal((*)a->value, p->p.flag.name))
            return 0;
        (*(int *)p->addr) = p->p.flag_def ^ 1;
        (*a)->used = 1;
        p->nelems++;
        return 1;
    case D_EXTERN:
        if (PrmExternFunc == NULL)
            return 0;
        /* get external data string */
        if ((extern_data = (*PrmExternFunc)(PR_FIELD, p->p.field)) == NULL)
            return 0;
        switch (*extern_data)
        case 'Y':

```

```

        (*(int *)p->addr) = 1;
        break;
    case 'N':
        (*(int *)p->addr) = 0;
        break;
    default:
        break;
    }
    return 1;
}
case D_DEFAULT:
    (*(int *)p->addr) = p->p.flag_def;
    return 1;
default:
    return 0;
}
}

int p_flag_help(p)
    prm_inst *p;
{
    char str[1024];
    sprintf(str, "%s [%c]", p->p.flag.name, (p->p.flag_def ? 'Y' : 'N'));
    print_help_line(str, NULL, p->comment);
}

int p_flag_destruct()
    /******
    /* free */
    /******
    static char *vp_free_format;
    static char *vp_free_def;
    int p_free_ident(b)
        char *b;
    {
        char *s;
        if ((s = match_string(b)) == NULL)
            return 0;
        vp_free_def = MatchStr;
        s = match_space(s);
        if ((s = match_format(s)) == NULL)
            return 0;
        vp_free_format = MatchStr;
        s = match_space(s);
        if (*s != '\0')
            return 0;
        return 1;
    }

    int p_free_print(p)
        prm_inst *p;

```

```

(
    char    target[256];

    elsetos(p->p.free.format, p->addr, target);
    fprintf(prmfp, "%s;\n", target);
)

int p_free_parse(b, ap, p)
char    *b;
va_list *ap;
prm_inst *p;
(
    char    *s, *e;

    p->p.free.def = vp_free_def;
    p->p.free.format = vp_free_format;

    return 1;
)

int p_free_get(p, flag, a)
prm_inst *p;
int    flag;
arg_inst **a;
(
    char    s[256];

    switch(flag)
    case D_FREE:
        if (p->nelems == 0)
            return 1;
            return 0;

    case D_ARG:
        prm_sscanf((*(a)->value, p->p.free.format, p->addr);
        (*(a)->used = 1;
        p->nelems++;
        return 1;

    case D_EXTERN:
        return 0;

    case D_DEFAULT:
        prm_sscanf(p->p.free.def, p->p.free.format, p->addr);
        return 1;
        default:
            return 0;
    }
)

int p_free_help(p)
prm_inst *p;
(
    char    str[1024];
    char    target[1024];
    void    *address;

    address = (void *) p->addr;
    prm_sscanf(p->p.free.def, p->p.free.format, address);
    elsetos(p->p.free.format, address, target);
)

```

```

    sprintf(str, "[%s]", target);

    print_help_line(str, p->p.free.format, p->comment);
)

int p_free_destruct()
/*****
static int prepare_argv(argv, s, delimiter)
char    ***argv;
char    *s;
char    *delimiter;
(
    int    argc;
    int    i;

    if (s == NULL)
        return 0;

    s = _prm_string_dup(s);
    argc = strtokcount(s, delimiter);
    *argv = (char **)realloc(NULL, sizeof(char *)*argc);

    (*argv)[0] = strtok(s, delimiter);
    for (i = 1; i < argc; i++)
        (*argv)[i] = strtok(NULL, delimiter);

    return argc;
)

static int strtokcount(s, delimiter)
char    *s;
char    *delimiter;
(
    int    c = 0;
    int    inword = 0;
    int    l = strlen(s);
    int    i;

    if (s == NULL)
        return 0;
    for (i = 0; i < l; i++)
        if (inword ^ (index(delimit, s[i]) == NULL)) {
            inword ^= 1;
            if (inword)
                c++;
        }

    return c;
)

static int elsetos(format, s, t)
char    *format,
        *t;
void    *s;
(
    char    *fmt;
    int    long_var;
    int    len;
)

```

```

while ((*format != '%') && (*format != '\0'))
    format++;
if (*format == '\0')
    return (0);
fmt = format;
format++;
long_var = 0;

while ((*format != ' ') && (*format != '\0')) {
    switch (*format) {
        case 'l':
            long_var = 1;
            break;
        case 'e':
        case 'f':
            if (long_var)
                sprintf(t, fmt, *((double *)s));
            else
                sprintf(t, fmt, *((float *)s));
            return (1);
        case 'h':
            sprintf(t, "%d", *((short *)s));
            return (1);
        case 'c':
            sprintf(t, fmt, *((char *)s));
            return (1);
        case 'o':
        case 'x':
        case 'd':
            if (long_var)
                sprintf(t, fmt, *((long *)s));
            else
                sprintf(t, fmt, *((int *)s));
            return (1);
        case 'M':
            t[0] = '\0';
            strcpy(t+1, *((char **)s));
            len = strlen(t);
            t[len] = '\0';
            t[len+1] = '\0';
            return (1);
        case 'S':
            t[0] = '\0';
            strcpy(t+1, s);
            len = strlen(t);
            t[len] = '\0';
            t[len+1] = '\0';
            return (1);
        case 's':
            t[0] = '\0';
            sprintf(t+1, fmt, s);
            len = strlen(t);
            t[len] = '\0';
            t[len+1] = '\0';
            return (1);
        case '0':
        case '1':
        case '2':
        case '3':
    }
}

```

```

case '4':
case '5':
case '6':
case '7':
case '8':
case '9':
    break;
default:
    return (0);
}
format++;

static equal(s1, s2)
char *s1;
char *s2;
{
    while (1) {
        if ((*s1 == '.') || (*s2 == '.'))
            return 1;
        if (*s1 != *s2)
            return 0;
        if (*s1 == '\0')
            return 1;
        s1++;
        s2++;
    }
}

static equain(s1, s2, n)
char *s1;
char *s2;
int n;
{
    int i = 0;

    while (1) {
        if (*s1 == '.')
            return 1;
        if (*s1 != *s2)
            return 0;
        if (*s1 == '\0')
            return 0;
        i++;
        if (i >= n)
            return 1;
        if (*s1 == '\0')
            return 1;
        s1++;
        s2++;
    }
}

static data_size(format)
char *format;
{
    int long_var;
    int ds;

    while ((*format != '%') && (*format != '\0'))
        format++;
}

```



```

if (*format == '\0')
    return (1);
format++;
long_var = 0;

while ((*format != '\0') && (*format != '\0')) {
    switch (*format) {
        case 'l':
            long_var = 1;
            break;
        case 'e':
        case 'f':
            if (long_var)
                ds = sizeof(double);
            else
                ds = sizeof(float);
            return(ds);
        case 'h':
            ds = sizeof(short);
            return (ds);
        case 'c':
            ds = sizeof(char);
            return (ds);
        case 'o':
        case 'x':
        case 'd':
            if (long_var)
                ds = sizeof(long);
            else
                ds = sizeof(int);
            return (ds);
        case 'M':
            ds = sizeof(char *);
            return (ds);
        case 'S':
        case 's':
            ds = PmStringSize;
            return (ds);
        case '0':
        case '1':
        case '2':
        case '3':
        case '4':
        case '5':
        case '6':
        case '7':
        case '8':
        case '9':
            break;
        default:
            return 0;
    }
    format++;
}

static string_copy(s1, s2, length)
char *s1;
char *s2;
int length;

```

```

{
    strncpy(s1, s2, length);
    s1[length] = '\0';
}

#ifdef MIN
#define MIN(a, b) ((a)<(b))?(a):(b)
#endif

static prm_sscanf(v, f, d)
char *v;
char *f;
void *d;
{
    switch (f[1]) {
        case 'M':
            if (d == NULL)
                return;
            *((char **)d) = (char *)realloc(NULL, strlen(v)+1);
            string_copy(*((char **)d), v, strlen(v));
            break;
        case 'S':
            string_copy(d, v, MIN(strlen(v), PmStringSize-1));
            break;
        default:
            sscanf(v, f, d);
    }
}

static print_help_line(def, fmt, comment)
char *def, *fmt, *comment;
{
    static int width[3] = {29, 6, 39};
    static char space[3][80] = {
        " ",
        " ",
        " "
    };
    char *fld[3];
    int len[3];
    int total = 0;
    char tmp[80];
    int i;

    fld[0] = def;
    fld[1] = fmt;
    fld[2] = comment;

    for (i = 0; i < 3; i++) {
        if (fld[i] == NULL)
            len[i] = 0;
        else
            len[i] = strlen(fld[i]);
        total += len[i];
    }

    while (total > 0) {
        for (i = 0; i < 3; i++) {

```

Sun Aug 9 10:42:57 1998

Listing for Adam Sartiel

```

if (len[i] == 0)
    fprintf(PrmFp, "%-*.*s", width[i], width[i],
            space[i]);
else if (len[i] <= width[i])
    fprintf(PrmFp, "%-*.*s", width[i], width[i],
            fld[i]);
total -= len[i];
len[i] = 0;
}
else {
    string_copy(tmp, fld[i], width[i]);
    fprintf(PrmFp, "%-*.*s\\", width[i], width[i],
            tmp);
total -= width[i];
fld[i] += width[i];
len[i] -= width[i];
}
}
putc('\\n', PrmFp);
}

char *_prm_string_dup(s)
char *s;
{
    char *t;
    if (s == NULL)
        return NULL;
    t = (char *)malloc(strlen(s)+1);
    string_copy(t, s, strlen(s));
    return t;
}

```



```
static char *rcsidd = "$Header: /tmp_mnt/home/users/gidi/util/lib/prm/RCS/prm.h,
v 1.3 92/07/23 16:55:30 gidi Exp Locker: gidi $";
```

```
/*
 * $Log: prm.h,v $
 * Revision 1.3 92/07/23 16:55:30 gidi
 * *** empty log message ***
 * Revision 1.2 91/11/15 14:57:12 gidi
 * new prm_argv header file
 */
```

```
/*
 * define parameter types
 */
```

```
#ifndef _prm_h_
#define _prm_h_

#ifdef FILE
#include <stdio.h>
#endif
```

```
#define ILLEGAL_LINE this is an illegal C line
```

```
/* flags for prm_argv mode argument */
```

```
#define P_HELP 1
#define P_PRINT 2
#define P_ERROR 4
#define P_IGNORE 8
#define EOLIST 0
#define FALSE 0
#define TRUE 1
```

```
/*
 *
 * parameters */
/*
 */
```

```
/*
 * parameter types :
 * 1. index - get value from a certain argument number
 * 2. field - get value from a "field=value" argument
 * 3. flag - get a boolean value from "flag" argument
 * 4. free - get value from a free argument
 */
```

```
enum _prm_id {PR_INDEX, PR_FIELD, PR_FLAG, PR_FREE, PR_MAX_PRM};
```

```
/* parameter type structure */
```

```
typedef struct _prm {
    char name[20]; /* parameter type name */
    int (*ident)(); /* identification function */
    int (*print)(); /* print result function */
}
```

prm.h

```
int (*parse)(); /* parse function */
int (*get)(); /* get parameter values function */
int (*help)(); /* print help function */
int (*destruct)(); /* destruct instance function */

) prm_type;
```

```
/*
```

```
 * get function arguments
 * 1. free - return number of free elements
 * 2. arg - get value from an argument instance
 * 3. extern - get value from external source
 * 4. default - get value from parameter default
 */
```

```
enum _get_type {D_FREE, D_ARG, D_EXTERN, D_DEFAULT};
```

```
int p_index_ident();
int p_index_print();
int p_index_parse();
int p_index_get();
int p_index_help();
int p_index_destruct();
```

```
int p_field_ident();
int p_field_print();
int p_field_parse();
int p_field_get();
int p_field_help();
int p_field_destruct();
```

```
int p_flag_ident();
int p_flag_print();
int p_flag_parse();
int p_flag_get();
int p_flag_help();
int p_flag_destruct();
```

```
int p_free_ident();
int p_free_print();
int p_free_parse();
int p_free_get();
int p_free_help();
int p_free_destruct();
```

```
typedef struct _p_index {
    int arg_index;
    char *format;
```

```
} p_index;
```

```
typedef struct _p_field {
    char *name;
    char *def;
    char *format;
    int maxlen;
    int elemsize;
    int *n_addr;
```

```
} p_field;
```

```
typedef struct _p_flag {
    char *name;
```

prm.h

```

    int    def;

} p_flag;

typedef struct _p_free {
    char    *def;
    char    *format;
} p_free;

/* parameter instance */

typedef struct _prm_inst {
    int    type;
    char    *comment;
    void    *addr;
    int    nelms;
    int    external;
    union {
        p_index    index;
        p_field    field;
        p_flag    flag;
        p_free    free;
    } p;
    struct _prm_inst *next, /* parameter specific data union */
        /* next instance in linked list */
        *prev; /* previous instance in linked list */
} prm_inst;

typedef struct _arg_inst {
    int    used;
    int    count;
    char    *value;
    struct _arg_inst *next, *prev;
} arg_inst;

#ifdef _prm_c

FILE *PrmFp = stderr;
int PrmStringSize = 80;
char *(*PrmExternFunc)() = NULL;

prm_type prm_types[] = {
    ("p_index", p_index_ident, p_index_print, p_index_parse,
     p_index_get, p_index_help, p_index_destruct),
    ("p_field", p_field_ident, p_field_print, p_field_parse,
     p_field_get, p_field_help, p_field_destruct),
    ("p_flag", p_flag_ident, p_flag_print, p_flag_parse,
     p_flag_get, p_flag_help, p_flag_destruct),
    ("p_free", p_free_ident, p_free_print, p_free_parse,
     p_free_get, p_free_help, p_free_destruct),
    ("", NULL, NULL, NULL, NULL, NULL, NULL)
};

char *MatchStr;

#else
extern FILE *PrmFp;
extern int PrmStringSize;
extern prm_type prm_types[];
extern char *MatchStr;
extern char *(*PrmExternFunc)();
#endif

```

```

#endif

```

```

OBS = prm.o prm_funcs.o match.o error.o try.o
SRCS = prm.c prm_funcs.c match.c error.c try.c
MAKEDPEND = makedepend
CHECKIN = ci
LIBS =
CFLAGS = -O2

```

```

try : $(OBS)
      $(CC) $(CFLAGS) $(OBS) -o $@ $(LIBS)

depend : $(SRCS)
          $(MAKEDPEND) $(CFLAGS) $(SRCS)

checkin : $(SRCS) Makefile
           $(CHECKIN) -l $?

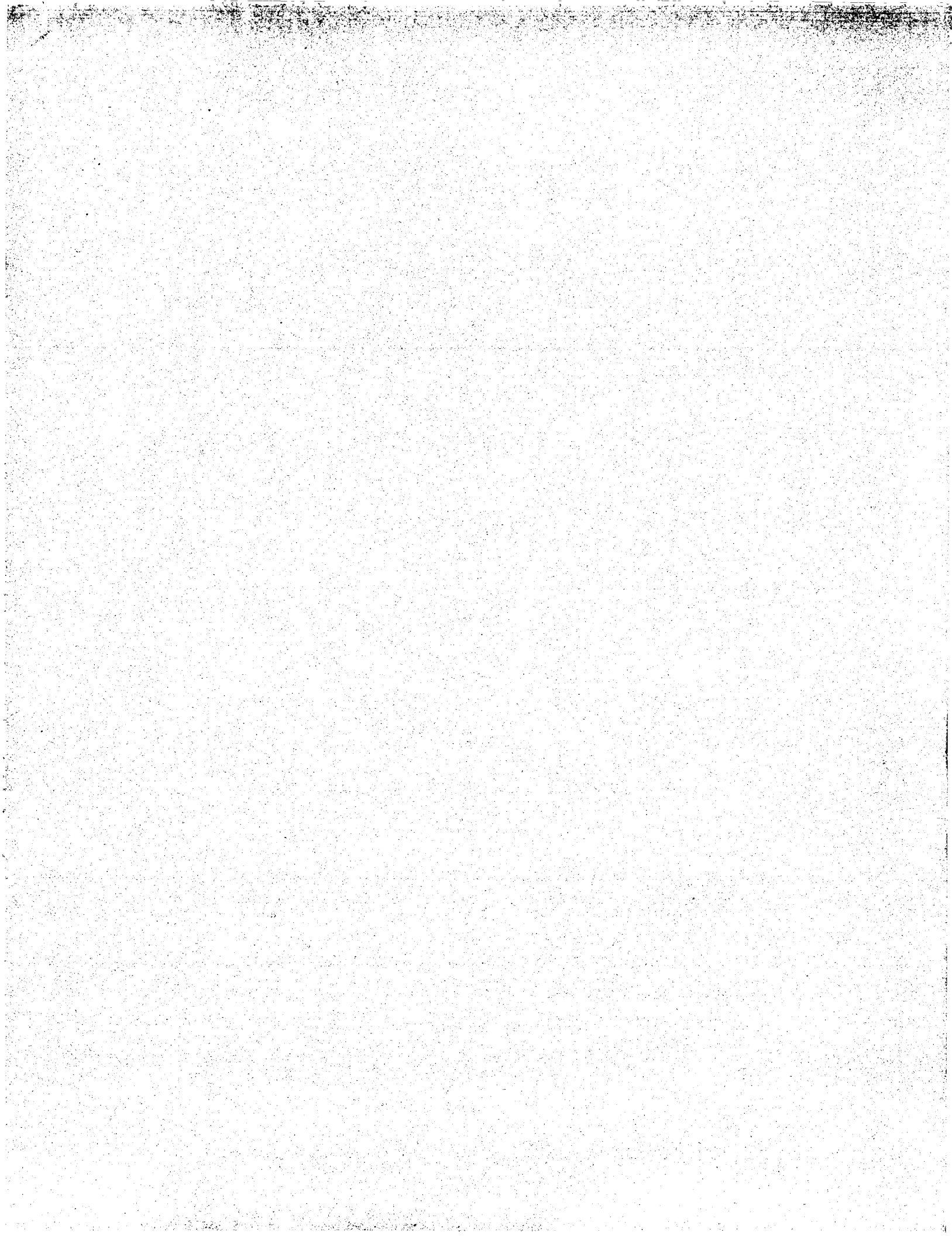
```

DO NOT DELETE THIS LINE -- make depend depends on it.

```

prm.o: /usr/include/varargs.h /usr/include/sys/va_list.h /usr/include/fcntl.h
prm.o: /usr/include/sys/types.h /usr/include/sys/feature_tests.h
prm.o: /usr/include/sys/isa_defs.h /usr/include/sys/machtypes.h
prm.o: /usr/include/sys/fcntl.h /usr/include/unistd.h
prm.o: /usr/include/sys/unistd.h /usr/include/ctype.h /usr/include/string.h
prm.o: prm.h /usr/include/stdio.h
prm_funcs.o: /usr/include/varargs.h /usr/include/sys/va_list.h
prm_funcs.o: /usr/include/ctype.h /usr/include/sys/feature_tests.h
prm_funcs.o: /usr/include/string.h prm.h /usr/include/stdio.h
match.o: /usr/include/ctype.h /usr/include/sys/feature_tests.h
error.o: /usr/include/stdio.h /usr/include/sys/feature_tests.h
error.o: /usr/include/sys/va_list.h /usr/include/varargs.h
try.o: prm.h /usr/include/stdio.h /usr/include/sys/feature_tests.h
try.o: /usr/include/sys/va_list.h

```



```
#!/usr/local/bin/perl -w
false=0;
true=1;
$debug=1;
use lib "/home/prod/lib/perl5/";
use Getopt::PmArgv;

require "USESYSYSTEM.pm";

#Utility for sorting big TEXT files with variable record length and number of s
ort keys
# gets the keys as regular expression and writes sort only the matches to the ke
ys

PmArgv("debug=0 &! debug level", \$debug,
"in= ./Clustering &! file to workmon", \$in,
"maxrec=0 &! number of records to sort", \$MaxRec,
"out=Unknown &! output file", \$out,
"-fasta -!", \$PastaFile,
"-check -!", \$JustCheck,
"-pad0 -!", \$PadZero,
"-unixsort -!", \$UnixSort,
"keysize=50 &! max key length", \$keyLength,
"keys=() &s *! ('regexp', 'regexp'...) list", \@keys);

$MaxRec=0 if ($MaxRec eq "");

print "MaxRec=\"\$MaxRec\"\n" if ($debug>2);

if (@keys == 0) {
    print "no keys to sort by!\n";
    exit 1;
}

#-----
#definitions

$PastaRecSep="\n>";
$RecSep=" ";
$RecSep=$PastaRecSep if ($PastaFile);
if ($RecSep eq " ") {
    print "no Record Separator defined, using fasta\n";
    $RecSep=$PastaRecSep;
}

$/ = $RecSep;
$BufLen=20;

$SepLen=length($/);
$PadChar=" "; # which character to use to separate key parts

if ($PadZero) {
    $PadChar=chr(0);
    print "padding with chr(0)\n" if ($debug>0);
}
```

```
print "the separator length is $SepLen\n" if ($debug>0);
print "using key length $keyLength\n" if ($debug>0);
#s* = 1;
$StartSeq=0;

$TempFile="keys".unsorted; #temp file to use if using unix sort
#-----
#readin keys and file pointers into a big hash

if ($JustCheck) {
    print "Checking that $in is sorted\n";
    CheckSorting();
    exit 0;
}

open (IN, $in) || die "CRITICAL ERROR: Query $query not found\n";
open (OUT, ">$out") || die "CRITICAL ERROR: cannot open output file $out";
if ($UnixSort) {
    open (TEMP, ">$TempFile") || die "CRITICAL ERROR: cannot open output file $TempF
ile";

    $RecJump=0;
    INPUT: {
        while (<IN>) {
            if ( ($RecJump == 100) && ($debug == 2) ) {
                print "working on record $. \n";
                $RecJump=0;
            }
            $RecJump++;

            $BigKey="";
            $len=length($_);
            s/^>//;
            s/^>//;
            foreach $CurKey (@keys) {
                $match="";
                print "testing $CurKey on \n $_" if ($debug>2);
                ($match) = $_CurKey/ms;
                $MatchLen=length($match);
                if ( $MatchLen == 0 ) {
                    print "No match to $CurKey\n";
                    exit 1;
                }
                if ( $MatchLen > $keyLength ) {
                    print "match is too long - $MatchLen working on keys 1-$keyLength\n";
                    exit 1;
                }

                print "found match was $match\n" if ($debug>2);
                #if we pad with chr(0) we dont need long padding
            }
        }
    }
}
```



```

if ($PadZero) {
    $match=$match.$padChar;
}
else {
    for ($i=$matchLen;$i<=$keyLength;$i++) {
        $match=$match.$padChar;
    }
}

$bigKey = $bigKey . "-" . $match;

}
$bigKey = $bigKey . "-" . $match;
print "final key was $bigKey\n" if ($debug>2);
$tempKey="$bigKey"."n";
print TEMP $tempKey if ($UnixSort);

$endSeq=tell(IN)-$seqLen;
$len=$len-$seqLen;
$len=$len-1 if (($i==1) && ($fastarFile)); # first record should be cut
$startSeq=$endSeq-$len;
$seq ($bigKey)=$i;
$start($i)=$startSeq;
$len($i)=$len;

print "The record starts in $startSeq and has $len char in it (without sep)
\n" if ($debug>2);
last INPUT if (($MaxRec > 0) && ($i == $MaxRec));
}

print "Worked on $. seq\n" if ($debug>1);
$len($i)=$len-1; # the last record is shorter - need to fix
close TEMP if ($UnixSort);
#-----

$RecordNum=(keys %Seq);
OUTPUT: {
    # if sort on disk needed
    if ($UnixSort) {
        USESYSTEM::ExecBinary("sort","InputFiles","OutputFile","sort $tempFile > keys
.sorted");
        open (TEMP, "keys.sorted") || die "CRITICAL ERROR: cannot open output file $
tempFile";
        $i=1;
        $i/="n"; #readin a \n seprated file
        while (<TEMP>){
            s/\n//;
            $Record-$_;
            print "read $_ doing key $Record\n" if ($debug >2);
            $startSeq=$start($Seq ($Record));
            $len=$len($Seq ($Record));
            PrintFinalRecord ($startSeq,$len,$i);
            $i++;
        }
        last OUTPUT;
    }
}

```

```

print "have $RecordNum records to work on \n" ;
$i=1;

# going over the records - sorting and printing

print "doing sort in memory \n" if ($debug>1);
foreach $Record ( sort keys %Seq) {
    $startSeq=$start($Seq ($Record));
    $CurLen=$len($Seq ($Record));
    print "going to print key \"$Record\" start=$startSeq len=$CurLen into outpu
t file \n" if ($debug>1);
    PrintFinalRecord ($startSeq,$CurLen,$i);
    $i++;
}

close (OUT);
close (IN);
#-----

sub PrintFinalRecord {
    my ($startSeq,$CurLen,$i)=( @_ );

    seek (IN,$startSeq,0);
    read (IN,$seq,$CurLen);
    print "===== \n" if ($debug>2);
    print " in seq \"($Record)\" start=$startSeq len=$CurLen seq=$seq\n" if ($debug
>2);
    $seq=$seq."$RecSep" if ($i < $RecordNum);
    $seq=$seq."n" if ($i == $RecordNum); # the last record needs an eol
    $seq->".$seq" if (($i == 1) && ($fastarFile));
    print OUT $seq;
}

#-----

sub CheckSorting {
    open (IN,$in) || die "CRITICAL ERROR: Query $query not found\n";
    $keyNum=1;
    foreach $CurKey (@keys) {
        $oldMatch($keyNum)="";
        $keyNum++;
    }
    $RecJump=0;
    while (<IN>) {
        if ( ($RecJump == 100) && ($debug == 2) ) {
            print "working on record $. \n";
            $RecJump=0;
        }
    }
}

```

```

$RecJump++;
$ok=0;
s/^>/;/;
s/^>/;/;
$KeyNum=1;

$KeyNum=1;
foreach $CurKey ($keys) {
    print "testing $CurKey on \n $_" if ($debug>2);
    ($match)= /$CurKey/ms;
    print "found $match , the one before was $oldMatch($KeyNum) key= $KeyNum\n"
    if ($debug>2);

    $MatchLen=length($match);
    if ( $MatchLen == 0 ) {
        print "No match to $CurKey \n";
        exit 1;
    }
    if ( $MatchLen > $KeyLength ) {
        print "match is too long - $MatchLen working on keys 1-$KeyLength\n";
        exit 1;
    }
    print "comapring $match with $oldMatch($KeyNum) \n" if ($debug>2);
    if (($match cmp $oldMatch($KeyNum)) == 1) {# new match is bigger lexicall
        y
        print "$match is bigger than $oldMatch($KeyNum) key :$KeyNum \n" if ($d
        ebug>2); $ok=1;
    }
    if ( (!($ok)) && ($match cmp $oldMatch($KeyNum)) == -1) {# new match is
    smaller lexically
        print "Found Mistake in sorting in line $. $match is smaller that $oldM
        atch($KeyNum)\n";
        exit 1;
    }
}

$oldMatch($KeyNum)=$match;
$KeyNum++;
}

print "File is O.K !\n";
}

```



```
***** Private Functions *****/
sub SplitClusterToLists(
    my ($MachinesNum)=0;
    my $i=0;
    # reading the seq.clu file
    my $rec=<IN>;
    $i=$rec;
    ($est,$cluster,$contig,$date,$estInCluster,$cc,$stam)=split ;
    $NumOfActiveLists=$MachinesNum;
    while(1){ # if there are more lines to read
        foreach $comp (keys %Machines){
            NEXT: {
                InitCounter() if ($NumOfActiveLists == 0);
                print "I still have $NumOfActiveLists lists to go on before zeroing \n"
                if ($debug > 2);
                last NEXT if ($Machines{$comp} == 0) ;
                if (!($rec)) {
                    print "finished $i lines\n";
                    exit 0;
                }
                print "writing contig $contig to $comp ".list \n" if ($debug > 2);
                print $comp "$cluster\.$contig\n";
                $Machines{$comp}--;
                if ($Machines{$comp} == 0) {
                    # one less list to work on
                    $NumOfActiveLists--;
                    print "decreasing lists to $NumOfActiveLists \n" if ($debug > 2);
                }
                $PrevContig=$contig;
                while ($contig eq $PrevContig){
                    print "still in contig $contig not writing\n" if ($debug > 2);
                    $rec=<IN>;
                    $i=$rec;
                    ($est,$contig,$cluster,$date,$estInCluster,$cc,$stam)=split ;
                }
            }
        }
    }
}
```

```
*****/
sub PrintComp {
    print "in PrintCounter\n";
    foreach $comp (keys %Machines){
        print "comp $comp power $Machines{$comp} \n";
    }
}

*****/
#parsing the config file
# return an array in which each entry is one
# line in the config file
sub ParseConfigFile{
    print "ParseConfigFile : opening $configfile\n" if ($debug>2);
    my ($configfile)=0;
    my @TempArray=();
    open (CONF , "<$configfile" || die "cant open $configfile";
    while (<CONF>) {
        PRINT : {
            s/^\\n//;
            last PRINT if ($_ eq "");
            last PRINT if (m/^\\#//);
            push @TempArray,($_);
        }
    }
    close (CONF);
    return reverse (@TempArray);
}
*****/
# initialize $Machines hash with the counters from the config file
sub InitCounter{
    print "InitCounter\n" if ($debug>4);
    foreach $comp (keys %Types){
        my $type=$Types{"$comp"};
        print "putting $typeAmount(\"$type\\") into \\$Machines\\{$comp\\} \n" if ($debug
        >4);
        $Machines{$comp}=$typeAmount("$type");
    }
    $NumOfActiveLists = $NumOfMachine;
}
*****/
#* End of SplitCluster2..pl */
#*
#*
```

```
#!/usr/local/bin/perl -w

#####
# * Name : SplitCluster2.pl
# * Purpose : split clusters list into machines lists according to computin a
#             bility
# * Author : Guy Kol (research)
# * Created : Jun 3 1998
# * Method : A script to create several lists of Cluster.Config from the seq.
#             clu file
#             in order to split assembly between several machines
#             reads a configuration file that lists the relative amount of clus
#             ter each architecture
#             should get ,and then a list of machines name and their architect
#             ures.
#             creates a "machines_name.list" file for each machine.
# *
# *
# *
# ***** global definitions *****/

use Shell;
use Getopt::PRMArgv;

PRMArgv("debug=0 %d! debug level",\$debug,
        "in= ./Clustering %s! file to workmon",\$in,
        "out_dir= %s! output file",\$out_dir,
        "config_file= %s! configuration file name ",\$configfile,
        "-check -f",\$justCheck);

# ***** Global definitions *****/
$NumOfMachine=0;

# ***** Global routines *****/

#checking config file
if (!(\$configfile)) {
    print "must have config file\n";
    exit 1;
}

print "checking configuration file \$configfile\n";
if (!(-r \$configfile)) {
    print "cannot read config file \$configfile\n";
    exit 1;
}

if (defined \$out_dir){
    print "checking \$out_dir\n";
    if (!(-x \$out_dir)) {
        print "cannot write to dir \$out_dir\n";
    }
}
```

SplitCluster.pl

```
exit 1;
}

# getting lines of config files

@LinesArray=ParseConfigFile(\$configfile);
print "LinesArray = @LinesArray\n" if (\$debug >2);
$LineCount=0;
while (@LinesArray){
    $LineCount++;
    $_=pop @LinesArray;
    if (m/^(([A-Z])+=(\d+))$/){
        print "found that $1=$2\n" if (\$debug >2);
        $typeAmount{"$1"}=$2;
    }
    if (m/^(([a-z0-9_]+)(\s+))$/){
        ($machine,$type)=split;
        print "found that $machine is $type\n" if (\$debug >2);
        $types{"$machine"}=$type;
        $NumOfMachine++;
    }
}

InitCounter();
PrintComp();

#
if ($NumOfMachine < 2) {
    print "has to be at least 2 machines\n";
    exit 1;
}

# checking we can open a list for each computer
open (IN, $in) || die "CRITICAL ERROR: input file $in not found\n";
$i=0;
foreach $comp (keys %Machines){
    my $fileName="$comp.list";
    $fileName="$out_dir/".$fileName if (defined $out_dir);
    print "writing to $fileName\n";
    open ($comp,"> $fileName") || die "cannot open $comp.list\n";

    #doing the split

    InitCounter();
    SplitClustersToLists($NumOfMachine);

    # cleaning

    close (IN);
    foreach $comp (keys %Machines){
        close $comp;
    }
}
```

SplitCluster.pl

```
#!/usr/local/bin/perl

# check command line syntax
if(@ARGV != 3){
    die "usage is : add_clone_info <clu file> <clone_info file> <out_file>;"
}

# get query
$in=shift(@ARGV);
$list=shift(@ARGV);
$out=shift(@ARGV);

open (LIST,$list) || die "CRITICAL ERROR: file $list not found\n";
while (<LIST>){
    $i++;
    ($name,$size,$token) = /\s+\s+(\s+)\s+(\s+)/;
    $name{$i} = $name;
    $size{$i} = $size;
    $fast{$name} = $i;
    $token{$i} = $token;
    $first{$token} = $i if (!first{$token});
    # print "$name{$i} $fast{$name{$i}} $token{$i} $first{$token{$i}}\n";
}
close (LIST);

open (IN,$in) || die "CRITICAL ERROR: file $in not found\n";
while (<IN>){
    ($name,$contig,$day,$month,$year,$size) =
        /\s+(\s+)\s+(\s+)\s+([0-9]+)-([0-9]+)-([0-9]+)-([0-9]+)\s+(\s+)/;
    # check if we have it in our list.
    if ($i = $fast{$contig}) {
        $token = $token{$i};
        $i = $first{$token};
        $j = 0;
        while ($token{$i} == $token) {
            # Do we already have a date associated with the contig ?
            if ($day{$i}) {
                $date = $year + $month/12 + $day/365;
                if ($date < $date{$i}) {
                    $day{$i} = $day;
                    $month{$i} = $month;
                    $year{$i} = $year;
                    $date{$i} = $date;
                    $cluster{$i} = $contig;
                }
            } else {
                $day{$i} = $day;
                $month{$i} = $month;
                $year{$i} = $year;
                $date{$i} = $year + $month/12 + $day/365;
                $cluster{$i} = $contig;
            }
            $j += $size{$i};
            $i++;
        }
        $size{$token} = $i - $first{$token};
        $size2{$token} = $j;
    }
}
```

```

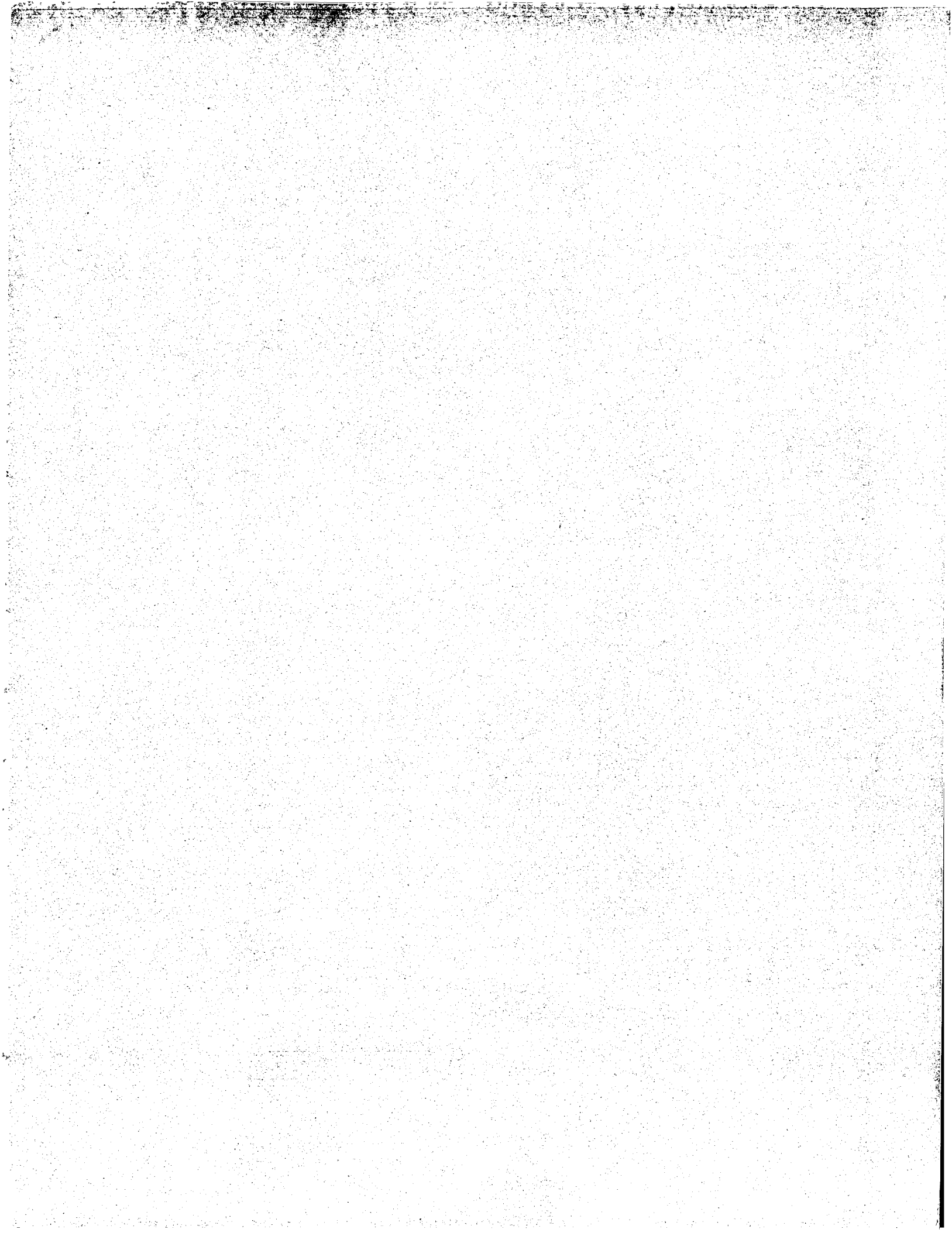
close (IN);

open (OUT, ">out") || die "CRITICAL ERROR: can't open output file\n";
open (IN, $in) || die "CRITICAL ERROR: file $in not found\n";
while (<IN) {
    ($name, $contig, $date, $size) = /\s*(\S+)\s*(\S+)\s*(\S+)\s*(\S+)*;/;
    $i = $fast{$contig};
    if ($i) {
        printf (OUT "%10s %10s %10s %s %6s %4d %6d\n", $name, $contig,
            $cluster[$i], $date, $size, $size{$token[$i]}, $size2{$token[$i]});
    } else {
        printf (OUT "%10s %10s %10s %s %6s %4d %6d\n", $name, $contig,
            $contig, $date, $size, 1, $size);
        #printf OUT "%name $contig $date $size\n";
    }
}
close (OUT);
close (IN);
}

```

add' clone info.pl

add clone info.pl



```
#!/usr/local/bin/perl

# check command line syntax
if (@ARGV != 4) {
    die "Usage is : clone_clusters.pl <rich_fasta_file> <max_size> <bridge> <output_file>\n";
}

# get query
$_ = shift(@ARGV);
$max_size = shift(@ARGV);
$_min_bridge = shift(@ARGV);
$_out = shift(@ARGV);

open (IN, $_) || die "CRITICAL ERROR: Query $query not found\n";
while (<IN>) {
    # We're interested only in header lines.
    if (/^>/) {
        # extract information
        ($clone) = /\#CLs+([^\s]+)/;
        ($cluster) = /\#CN\s+CC[HM][1-9][0-9][0-9]_([0-9])_([^\s]+)/;
        ($size) = /\#SZ\s+([^\s]+)/;
        if ($size <= $max_size) {
            # print "$clone $cluster $size\n";
            # save the sizes of the clusters
            $sizes{$cluster} = $size;
            $cl_no++ if ($cluster ne $old_cluster);
            $old_cluster = $cluster;

            # create a hash of clone links
            if ($clone) {
                $times{$clone}++;
                $cl1{$clone} = $cluster if ($times{$clone} == 1);
                $cl2{$clone} = $cluster if ($times{$clone} == 2);
            }
        }
    }
}
close (IN);
print "Read $cl_no sequences with clone information, created first hash\n";

# Go over the pairs, create the second hash
foreach $clone (keys %times) {
    # use only clones which appear exactly twice.
    if ($times{$clone} == 2) {
        # create an index composed of the two cluster names (sorted).
        if ($cl1{$clone} lt $cl2{$clone}) {
            $index = $cl1{$clone} . " " . $cl2{$clone};
        } else {
            $index = $cl2{$clone} . " " . $cl1{$clone};
        }
        # count the number of times the pair appeared.
        $link{$index}++;
    }
}
print "Created second hash\n";

open (OUT, ">$out") || die "CRITICAL ERROR: can't open output file\n";
foreach $index (keys %link) {
```

```
($first, $second) = ($index =~ /\s+\s+(\S+)/);
if ($first ne $second) {
    $bridge = $link{$index};
    if ($bridge >= $_min_bridge || $bridge == $sizes{$second}) {
        print OUT "$index $sizes{$first} $sizes{$second}\n";
    }
}
close (OUT);
```


Listing for Adam Santel

Sun Aug 9 10:45:42 1998

```
#!/usr/local/bin/perl

# check command line syntax
if (@ARGV != 2) {
    die "\nusage is : clone_clusters.pl <rich_fasta_file> <out_file>\n";
}

# get query
$in = shift(@ARGV);
$out = shift(@ARGV);

# go over the pairs, do the union-find
open (IN, $in) || die "CRITICAL ERROR: can't open input file\n";
open (OUT, ">$out") || die "CRITICAL ERROR: can't open output file\n";
while (<IN>) {
    $pair_no++;
    ($first, $second, $s1, $s2) = /(\\S+)\\s+(\\S+)\\s+(\\S+)\\s+(\\S+)/;

    # add "first" to list (if not there)
    if (!$pos{$first}) {
        $spos{$first} = $s1;
        $depth{$s1} = 1;
        $fathers{$s1} = $s1;
        $sizes{$first} = $s1;
    }

    # add "second" to list (if not there)
    if (!$pos{$second}) {
        $spos{$second} = $s2;
        $depth{$s2} = 1;
        $fathers{$s2} = $s2;
        $sizes{$second} = $s2;
    }

    # find the ancestor of "first"
    $ind1 = $ind = $pos{$first};
    do {
        $ind1 = $fathers{$ind1};
    } while ($ind1 != $fathers{$ind1});
    # compress path
    do {
        $tmp = $fathers{$ind};
        $fathers{$ind} = $ind1;
        $ind = $tmp;
    } while ($ind != $ind1);

    # find the ancestor of "second"
    $ind2 = $ind = $pos{$second};
    do {
        $ind2 = $fathers{$ind2};
    } while ($ind2 != $fathers{$ind2});
    # compress path
    do {
        $tmp = $fathers{$ind};
        $fathers{$ind} = $ind2;
        $ind = $tmp;
    } while ($ind != $ind2);

    # find deeper cluster
    if ($depth{$ind1} < $depth{$ind2}) {
        $tmp = $ind1; $ind1 = $ind2; $ind2 = $tmp;

        # do the union
        $depth{$ind2} = $ind1;
        $depth{$ind1}++; if ($depth{$ind1} == $depth{$ind2});
    }
    $cl_no = $i;
    print "Finished union-find of $pair_no pairs.\n";
    for ($i = 1; $i <= $cl_no; $i++) {
        $ind = $i;
        do {
            $ind = $fathers{$ind};
        } while ($ind != $fathers{$ind});
        $fathers{$i} = $ind;
    }

    print "Fixed cluster numbers.\n";

    foreach $cluster (sort by_cluster keys %pos) {
        print OUT "$cluster $sizes{$cluster} $fathers{$pos{$cluster}}\n";
    }
    close (OUT);

    print "finished printing. Done.\n";

    sub by_cluster { $fathers{$pos{$a}} <=> $fathers{$pos{$b}} };
}
```

clone_clusters2.pl

Listing for Adam Santel

Sun Aug 9 10:45:42 1998

```
#!/usr/local/bin/perl

# check command line syntax
if (@ARGV != 2) {
    die "\nusage is : clone_clusters.pl <rich_fasta_file> <out_file>\n";
}

# get query
$in = shift(@ARGV);
$out = shift(@ARGV);

# go over the pairs, do the union-find
open (IN, $in) || die "CRITICAL ERROR: can't open input file\n";
open (OUT, ">$out") || die "CRITICAL ERROR: can't open output file\n";
while (<IN>) {
    $pair_no++;
    ($first, $second, $s1, $s2) = /(\\S+)\\s+(\\S+)\\s+(\\S+)\\s+(\\S+)/;

    # add "first" to list (if not there)
    if (!$pos{$first}) {
        $spos{$first} = $s1;
        $depth{$s1} = 1;
        $fathers{$s1} = $s1;
        $sizes{$first} = $s1;
    }

    # add "second" to list (if not there)
    if (!$pos{$second}) {
        $spos{$second} = $s2;
        $depth{$s2} = 1;
        $fathers{$s2} = $s2;
        $sizes{$second} = $s2;
    }

    # find the ancestor of "first"
    $ind1 = $ind = $pos{$first};
    do {
        $ind1 = $fathers{$ind1};
    } while ($ind1 != $fathers{$ind1});
    # compress path
    do {
        $tmp = $fathers{$ind};
        $fathers{$ind} = $ind1;
        $ind = $tmp;
    } while ($ind != $ind1);

    # find the ancestor of "second"
    $ind2 = $ind = $pos{$second};
    do {
        $ind2 = $fathers{$ind2};
    } while ($ind2 != $fathers{$ind2});
    # compress path
    do {
        $tmp = $fathers{$ind};
        $fathers{$ind} = $ind2;
        $ind = $tmp;
    } while ($ind != $ind2);

    # find deeper cluster
    if ($depth{$ind1} < $depth{$ind2}) {

```

clone_clusters2.pl

complete_rich_fasta.pl

Sun Aug 9 10:45:42 1998

Listing for Adam Santiel

```
#!/usr/local/bin/perl

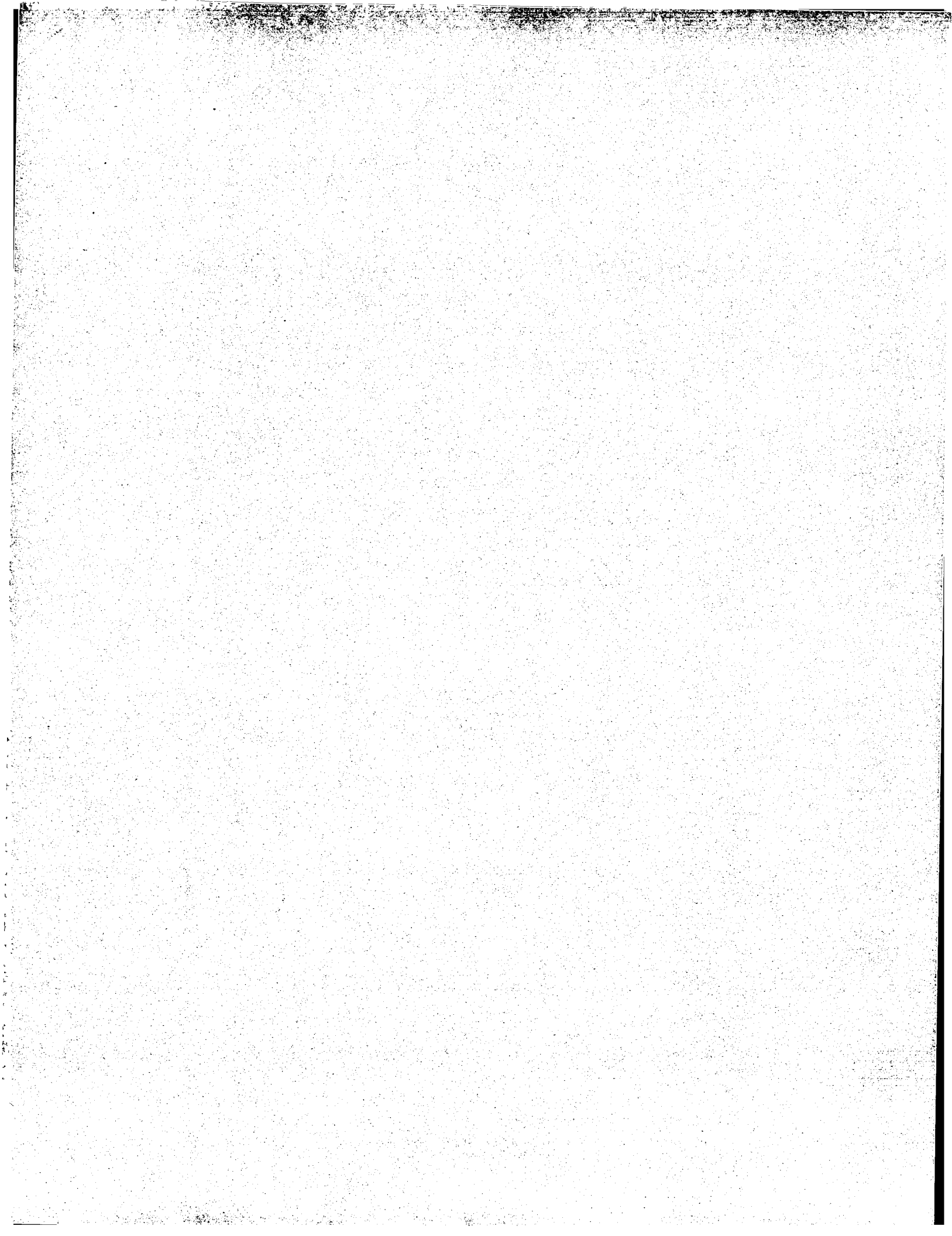
# check command line syntax
if (@ARGV != 3) {
    die "usage is : delete_big_pairs.pl <list_file> <pairs_file> <out_file>";
}

# get query
$list=shift(@ARGV);
$in=shift(@ARGV);
$out=shift(@ARGV);

open (LIST, $list) || die "CRITICAL ERROR: file $list not found\n";
while (<LIST>) {
    ($name) = /\s+/;
    $bit($name) = 1;
}
close (LIST);

open (OUT, ">$out") || die "CRITICAL ERROR: can't open output file\n";
open (IN, $in) || die "CRITICAL ERROR: file $in not found\n";
while (<IN>) {
    ($name1, $name2) = /\s+\s+\s+/;
    $sum = $bit($name1) + $bit($name2);
    print OUT $_ if ($sum == 0);
    print "Error! : $_" if ($sum == 1);
}
close (IN);
close (OUT);
```

delete_big_pairs.pl



```
#!/usr/local/bin/perl
use Getopt::Std;
use StructuredFile;
use integer;

PmArgv("in= clusters.EMBL $input file name", \infile,
"outprefix= clusters.embl $ioutput files prefix name", \outpref,
"max_size= 200000000 $dmaximum size of each output file", \smax_size,
"file_type= embl $sfile type (for structuredfile)", \sfile_type);

# calculate average size of each output file
if ( ! (-e $infile) )
{
    die "$infile doesn't exist - aborting.\n";
}
#bug if ( ! ($dev,$ino,$mode,$nlink,$uid,$gid,$rdev,$size,
# $atime,$mtime,$ctime,$blksize,$blocks)
# = stat(_) ) )
# {
#     die "couldn't get stat for $infile - aborting.\n";
# }
# use open, seek and tell to get file's size... bug in stat
open (TRY, $infile) || die "couldn't open file $infile.\n";
seek TRY, 0, 2;
$ssize = tell TRY;

$num_of_files = $size/$smax_size + 1;
$avg_size = $size / $num_of_files;
if ( $avg_size == 0 )
{
    $avg_size = $smax_size;
}

# read file and split it up
$input = new StructuredFile('name' => $infile,
'type' => $file_type);
$i_output = 1;

open (OUTPUT_FILE, ">" . $outpref . $i_output) ||
    die "couldn't open output file $outpref$i_output - aborting.\n";
$i_output = 0;

while ( ! $input->eof() )
{
    ($rec, $id) = $input->get_record;
    $l_rec = length $rec;

    # fits in the current averaged sized file
    if ( $l_output + $l_rec < $avg_size )
    {
        print OUTPUT_FILE $rec || die "record:\n$rec\nfailed adding to file - ab
```

```
orting\n";
    $l_output += $l_rec;
}

# doesn't fit in current file (not even in the maximum allowed)
elseif ( $l_output + $l_rec > $smax_size )
{
    close OUTPUT_FILE;

    # recalc average size
    $ssize -= $l_output;
    $num_of_files = $size/$smax_size + 1;
    $avg_size = $size / $num_of_files;
    if ( $avg_size == 0 )
    {
        $avg_size = $smax_size;
    }

    $i_output++;
    open (OUTPUT_FILE, ">" . $outpref . $i_output) ||
        die "couldn't open output file $outpref$i_output - aborting.\n";
    $l_output = 0;

    # doesn't fit in ANY file allowed?
    if ( $l_rec > $smax_size )
    {
        die "record:\n$rec\n is larger than allowed size ($smax_size) - abort
ing.\n";
    }
    print OUTPUT_FILE $rec || die "record:\n$rec\nfailed adding to file - ab
orting\n";
    $l_output += $l_rec;
}

# fits in current file (but a little more than the average size)
else
{
    print OUTPUT_FILE $rec || die "record:\n$rec\nfailed adding to file - ab
orting\n";
    close OUTPUT_FILE;

    # recalc average size
    $ssize -= $l_output;
    $num_of_files = $size/$smax_size + 1;
    $avg_size = $size / $num_of_files;
    if ( $avg_size == 0 )
    {
        $avg_size = $smax_size;
    }

    $i_output++;
    open (OUTPUT_FILE, ">" . $outpref . $i_output) ||
        die "couldn't open output file $outpref$i_output - aborting.\n";
    $l_output = 0;
}
}
```

Listing for Adam Sartiel

Sun Aug 9 10:45:42 1998

```
close OUTPUT_FILE;  
print "Ended - split into $i_output files.\n";
```

divide_file.pl

```

#!/usr/local/bin/perl -w

# check command line syntax
if($ARGV != 3 && $ARGV != 4){
    die "usage is : $0 <query_file(formatted)> <db_file> <out_file> [e_score]";
}

# get query
$in=shift(@ARGV);
$db=shift(@ARGV);
$out=shift(@ARGV);
$e_score=shift(@ARGV) || 0.0001;

# Command lines for BLAST
#####
# /godiva1/cluster/exe/blast2/formatdb -i mouse_rep -p F
# /godiva1/cluster/exe/blast2/blastall -p blastn -d _ -i _ -
# -G 11 -E 22 -q -48 -r 10 -v 0 -b 1000 -e 0.0001 -o _

open (IN, "$db") || die "Can't open db file\n";
open (OUT, ">$out.unsigned") || die "Can't open output file\n";
while (<IN>) {
    s/^>//m;
    s/^>>//m;
    ($name) = />(\S+)\s/m;
    $name = `tr/a-zA-Z0-9_/_/c`;
    s/>>//m;
    open (SEQ, ">$out.$ENV(HOST).$.tmpseq") || die "Can't create file $out.$ENV(HOST).$.tmpseq\n";
    print SEQ $name;
    close (SEQ);
    print "Doing BLAST with $name...\n";
    system "blastall -p blastn -i $out.$ENV(HOST).$.tmpseq -d $in -v 5 -e $e_score -b 30000 -o $name.tmp";
    if ($? != 0) {
        system "/bin/rm $out.$ENV(HOST).$.tmpseq";
        die "Error running blastall!\n";
    }
    print "Done. parsing output...\n";
    open (IN2, "$name.tmp") || die "Can't open BLAST output file\n";
    $mode = "Start";
    $line = 0;
    while (<IN2>) {
        $line++;
        if (/^Query= (\S+)/) {
            if ($mode eq "Possible End") {
                if ($l1 < $l2) {
                    print OUT "$query $hit $sign $score $l1 $l2\n";
                } else {
                    print OUT "$query $hit $sign $score $l2 $l1\n";
                }
            }
            ($query) = $l1;
            $mode = "Got query";
        }
        if (/^>(\S+)/) {

```

```

if ($mode eq "Possible End") {
    if ($l1 < $l2) {
        print OUT "$query $hit $sign $score $l1 $l2\n";
    } else {
        print OUT "$query $hit $sign $score $l2 $l1\n";
    }
} else {
    print "ERROR! Got a hit with no query at line $line\n"
        if ($mode ne "Got query");
    ($hit) = $l;
    $mode = "Got hit";
}
if ($s+Score = $s+$s+bits/$s*((($s+)))/) {
    print "ERROR! Got a score with no hit at line $line\n"
        if ($mode ne "Got hit" && $mode ne "Possible End");
    if ($mode eq "Possible End") {
        if ($l1 < $l2) {
            print OUT "$query $hit $sign $score $l1 $l2\n";
        } else {
            print OUT "$query $hit $sign $score $l2 $l1\n";
        }
    }
    ($score) = $l;
    $first_line = 1;
    $mode = "Got score";
}
if (/^Query:\s+([0-9]*)[^\s]*([0-9]*)$/) {
    ($start, $end) = ($1, $2);
    if ($mode eq "Possible End" || $mode eq "Got score") {
    } else {
        print "ERROR! Got query info in mode $mode at line $line\n";
        $mode = "Query info";
    }
}
if (/^Sbjct:\s+([0-9]*)[^\s]*([0-9]*)$/) {
    ($start, $end) = ($1, $2);
    if ($mode eq "Query info") {
        $l1 = $start-1 if ($first_line);
        $l2 = $end-1;
        $sign = '+';
        $sign = '-' if ($start > $end);
    } else {
        print "ERROR! Got Subject info in mode $mode at line $line\n";
        $first_line = 0;
        $mode = "Possible End";
    }
}
if ($mode eq "Possible End") {
    if ($l1 < $l2) {
        print OUT "$query $hit $sign $score $l1 $l2\n";
    } else {
        print OUT "$query $hit $sign $score $l2 $l1\n";
    }
}
close (IN2);
$/ = "\n>";
}

```

do blast c/n pl

do blast clin-pl

```
# $* = 1;
} print "Done.\n";
close (IN);
close (OUT);

system "/bin/rm $out.$ENV{HOST}.$$tmpseq";

system "sort +1 $out.unsorted > $out";
if ( $? == 0 ) {
    print "removing $out.unsorted\n";
    system "/bin/rm $out.unsorted";
}
else {
    die "Couldn't sort $out.unsorted\ntry \"sort +1 $out.unsorted >$out\" yourse
lf \n";
}
```

do_blast_cln.pl

A-557

```
#!/usr/local/bin/perl
use Getopt::Std;

PmArgv("-show -!",$show,
"debug=0 &d! debug level",$debug,
"in=",$sfile to workmon,$gen_in,
"-append -lappend the output",$append,
"out=",$s! output file",$outfile,
"organism=",$s! organism,$organism,
"molecule=",$s! est/rna,$molecule);

# check command line syntax

if ($organism =~ /[hH]uman/){
    $organism="H";
}
if ($organism =~ /[mM]ouse/){
    $organism="M";
}
if ($molecule eq "est"){
    $molecule="E";
}
if ($molecule eq "rna"){
    $molecule="R";
}

print "got organism = $organism molecule=$molecule\n" if ($debug);
if ($organism eq "M"){
    $orgn = "Mus musculus";
    print "\nThis script will extract mouse ";
    if ($molecule eq "E"){
        print "EST's\n";
    }
    elsif ($molecule eq "R"){
        print "RNA's\n";
    }
}
else{
    print "\nWrong! the correct usage is: fetch.pl ";
    print "[MH] [ER] gen-in gen-out fasta-out\n";
    exit;
}

)
elsif ($organism eq "H"){
    $orgn = "Homo sapiens";
    print "\nThis script will extract human ";
    if ($molecule eq "E"){
        print "EST's\n";
    }
    elsif ($molecule eq "R"){
        print "RNA's\n";
    }
}
else{
    print "\nWrong! the correct usage is: nextextract.pl ";
    print "[MH] [ER] gen-in gen-out fasta-out\n";
    exit;
}

)
else {
    print "\nWrong! the correct usage is: nextextract.pl ";
    print "[MH] [ER] gen-in gen-out fasta-out\n";
    exit;
}
}
```

fetch.pl

```
open (GEN_IN, $gen_in) || die "CRITICAL ERROR: cannot open infile $gen_in\n";

if (($append) && (!f $outfile)) { # add results to existing file
    open (OUT, ">>$outfile") || die "CRITICAL ERROR: cannot open outfile $outfi
le\n";
}
else {
    open (OUT, ">$outfile") || die "CRITICAL ERROR: cannot open outfile $outfi
le\n";
}
# open (GEN_OUT, ">$gen_outfile");

select((select(OUT), $|=1)[0]);
$num=0;

$complete=0;
$RecNum=0;
while(<GEN_IN){
    $line[$num] = $_; # Inserting the input lines into the $line[] array.
    $num++;

    if ($molecule eq "R"){
        if ((/^LOCUS/)&& (/RNA/)) {
            if (($temp_name, $temp_length) = (/^LOCUS\s+(\S+)\s+(\d+)\s+/)) {
                $length=$temp_length;
                $name=$temp_name;
                $quality=0;
                $org=0;
                $bad_trace=0;
                $c_num="";
                $lib="";
                $tissue="";
                $clone="";
                $insert="";
                $nid="";
                $acc="";
                $def="";
                $seq="";
                $dev="";
                $sex="";
                $dir="";
                ($temp_date)=(/(\s+(\S+)\s+)$/);
            }
        }
        if ($molecule eq "E"){
            if ((/^LOCUS/)&& (/EST/)) {
                if (($temp_name, $temp_length) = (/^LOCUS\s+(\S+)\s+(\d+)\s+/)) {
                    $fetch=1;
                    $length=$temp_length;
                    $name=$temp_name;
                    $quality=0;
                    $org=0;
                    $bad_trace=0;
                    $c_num="";
                    $lib="";
                    $tissue="";
                    $clone="";
                }
            }
        }
    }
}
```

fetch.pl

```

$insert="";
$nid="";
$sacc="";
$sdef="";
$seq="";
$sdev="";
$ssex="";
$dir="";
($temp_date)=(/s+(\S+$/);

)

)

)
if ($ORGANISM eq "M"){
    if ((/^\s+ORGANISM\s+[mM]\s+/)&&
        ((/s+[Dd]omesticus\s*$/)||(/s+[Mm]usculus\s*$/)))
        $org=1;
    }
} else {
    if ((/^\s+ORGANISM\s+/)&&(/[Hh]omo\s+[Ss]apiens\s*$/))
        $org=1;
    }
}

if (/^NID/) {
    ($nid)=/NID\s+(\S+$/);
}
if (/^DEFINITION/) {
    ($def)=/^DEFINITION\s+(.+)/;
    $a=$num;
}
if (/^ACCESSION/) {
    ($sacc)=/^ACCESSION\s+(\w+)/;
    if ($num==$a+1){
    }
    elseif ($num==$a+2){
        $line[$num-2]=~/s+(.+)/;
        $def=$def." $1";
    }
    elseif ($num==$a+3){
        $line[$num-3]=~/s+(.+)/;
        $def=$def." $1";
        $line[$num-2]=~/s+(.+)/;
        $def=$def." $1";
    }
    elseif ($num==$a+4){
        $line[$num-4]=~/s+(.+)/;
        $def=$def." $1";
        $line[$num-3]=~/s+(.+)/;
        $def=$def." $1";
        $line[$num-2]=~/s+(.+)/;
        $def=$def." $1";
    }
}
if ((/[Hh]igh\s+quality/)) {
    ($quality)=/(\d+)/;
}
if ([/Tt]race/s&/poor/) {
    $bad_trace=1;
}

```

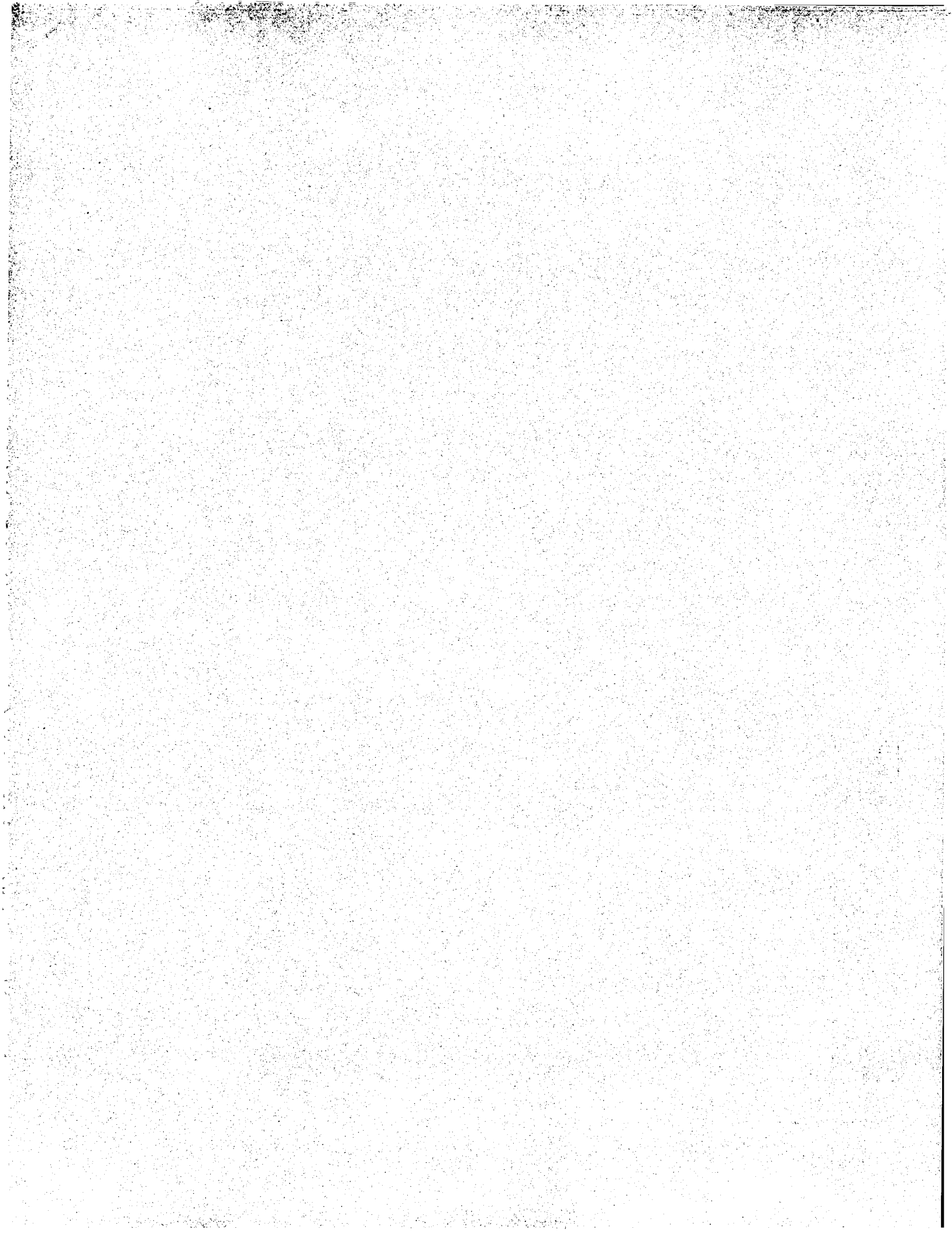
```
%
if (($temp_sex) = /sex=s*(.+)\*/) {
    $sex=$temp_sex;
}
if (($temp_tissue) = /tissue_type=s*(.+)\*/) {
    $tissue=$temp_tissue."T $temp_tissue ";
}
if (($temp_line) = /cell_line=s*(.+)\*/) {
    $tissue=$tissue."L $temp_tissue ";
}
if (($temp_stage) = /dev_stage=s*(.+)\*/) {
    $stage=$temp_dev;
}
if ($clone_lib = /clone_lib=s*(.+)\*/) {
    $lib=$temp_lib;
}
if (($temp_ins) = /insert Length: \s+(\d+)\/) {
    $insert=$temp_ins;
}
if (/[Ch]romosome/) {
    temp_C_num="";
    if (/([ch])romosome\s*\") {
        $(temp_C_num)=/\\"(\S+\\s*\$*\$*)\\""/;
        C_num=temp_C_num." " . $temp_C_num;
    } else {
        $(temp_C_num)/=romosome[\s+=]\(\\S+\\s*\$*\$*/);
        C_num=temp_C_num." " . $temp_C_num;
    }
}
if ($(temp_C_num)!=/\\S+\)[Cc]romosome-linked/) {
    C_num=temp_C_num." " . $temp_C_num;
}
if (/^\\s*[Cc]romosome-linked/) {
    temp_line=$_;
    $_=$line[$_num];
    $(temp_C_num)=/(\\S+)\\s*$/;
    C_num=temp_C_num." " . $temp_C_num;
    $_=$temp_line;
}
}
if (($temp_clone) = /clone=s*(.+)\*/) {
    $clone=$temp_clone;
}
}
if (/^ORIGIN/) {
    seq="";
    read_seq;
}
}
if (/^\\\/) {
    if ($fetch=1) {
        print "\nWORKING ON -- $name";
        fetch=0;
        if ($orig==1) {
            # End of "file".

```

fetch.pl

fetch.pl

fetch.pl



```
#!/usr/local/bin/perl

# check command line syntax
if (@ARGV != 3 && @ARGV != 4) {
    die "usage is : get_ests <list_file> <fasta_file> <out_file> [<out2_file>]";
}

# get query
$list=shift(@ARGV);
$in=shift(@ARGV);
$out=shift(@ARGV);
if (@ARGV) {
    $out2=shift(@ARGV);
    $flag = 1;
}

open (LIST, $list) || die "CRITICAL ERROR: file $list not found\n";
while (<LIST>) {
    ($name) = /(\\S+)/;
    $names{$name} = 1;
}
close (LIST);

open (OUT, ">$out") || die "can't open output file\n";
open (OUT2, ">$out2") || die "can't open second output file\n" if ($flag);
open (IN, $in) || die "CRITICAL ERROR: file $in not found\n";
$/ = "\n";
$* = 1;
while (<IN>) {
    s/^>///;
    s/^>///;
    ($name) = />(\\S+)\\s/;
    print OUT $name if ($names{$name} == 1);
    print OUT2 $_ if ($names{$name} != 1 && $flag);
}
close (IN);
close (OUT);
```



```
#!/usr/local/bin/perl

# check command line syntax
if (@ARGV != 5) {
    die "usage is : make_fasta.pl <list_file> <fasta_file1> <fasta_file2> \n-pre
fix> <out_file>";
}

# get query
$list=shift(@ARGV);
$in1=shift(@ARGV);
$in2=shift(@ARGV);
$prefix=shift(@ARGV);
$out=shift(@ARGV);

open (LIST, $list) || die "CRITICAL ERROR: file $list not found\n";
while (<LIST>) {
    ($name, $cluster, $size) = /(\\S+)\\s+(\\S+)\\s+(\\S+)/;
    $cluster{$name} = $cluster;
    $size{$cluster} = $size;
}
close (LIST);

open (OUT, ">$out") || die "CRITICAL ERROR: can't open output file\n";
open (IN, $in1) || die "CRITICAL ERROR: file $in1 not found\n";
$* = 1;
while (<IN>) {
    s/^>///;
    ($name) = /(\\S+)\\s/;
    s/>$/;/;
    $cluster = $cluster{$name};
    $size = $size{$cluster};
    s/^\\S+//;
    print OUT ">$name \\#CN $prefix$cluster \\#SZ $size $_" ;
}
close (IN);

open (IN, $in2) || die "CRITICAL ERROR: file $in2 not found\n";
while (<IN>) {
    s/^>///;
    ($name) = /(\\S+)\\s/;
    s/>$/;/;
    $cluster = $cluster{$name};
    $size = $size{$cluster};
    s/^\\S+//;
    print OUT ">$name \\#CN $prefix$cluster \\#SZ $size $_" ;
}
close (IN);
close (OUT);
```



```
#!/usr/local/bin/perl
use Getopt::Long;
use StructuredFile;
use integer;

# This script gets a list of files, and combines them into a number
# of files, each one no more than "max_size" bytes (well, maybe sometimes
# a little more...).

PmArgv("outprefix= clusters.embl %s!output files prefix name", \soutpref,
"max_size= 200000000 %s!maximum size of each output file", \smax_size,
"file_type= embl %s!file type (for structuredfile)", \sfile_type);

# calculate average size of each output file
$names = ();
$#names = $#ARGV;
$size = 0;

$ifile = 0;
while ( $ifile <= $#names ) {
    $names[$ifile] = $ARGV[$ifile];
    if ( ! ( -e $names[$ifile] ) ) {
        die "Names[$ifile] doesn't exist - aborting.\n";
    }
}

# bug if ( ! ( ($dev,$ino,$mode,$nlink,$uid,$gid,$rdev,$size,
#             $atime,$mtime,$ctime,$blksize,$blocks)
#             = stat(_) ) )
# {
#     die "couldn't get stat for $names[$ifile] - aborting.\n";
# }
# use open, seek and tell to get file's size... bug in stat

open (TRY, $names[$ifile]) || die "couldn't open file $names[$ifile].\n";
seek TRY, 0, 2;
$size += tell TRY;
close TRY;
$ifile++;
}

$num_of_files = $size/$max_size + 1;
$avg_size = $size / $num_of_files;
if ( $avg_size == 0 )
{
    $avg_size = $max_size;
}

# read files and split them up
$ifile = 1;
open (OUTPUT_FILE, ">" . $outpref . $ifile) ||
```

```
die "couldn't open output file $outpref.$ifile - aborting.\n";
$ifile = 0;
while ( $ifile <= $#names ) {
    $input = new StructuredFile('name' => $names[$ifile],
                                'type' => $file_type);

    while ( ! $input->eof() ) {
        ($rec, $id) = $input->get_record;
        $l_rec = length $rec;

        # fits in the current averaged sized file
        if ( $l_output + $l_rec <= $avg_size )
        {
            print OUTPUT_FILE $rec || die "record:\n$rec\nfailed adding to file
            - aborting.\n";
            $l_output += $l_rec;
        }

        # doesn't fit in current file (not even in the maximum allowed)
        elsif ( $l_output + $l_rec > $max_size )
        {
            start_new_file();

            # doesn't fit in ANY file allowed?
            if ( $l_rec > $max_size )
            {
                die "record:\n$rec\n is larger than allowed size ($max_size) - a
                borting.\n";
            }

            print OUTPUT_FILE $rec || die "record:\n$rec\nfailed adding to file
            - aborting.\n";
            $l_output += $l_rec;
        }

        # fits in current file (but a little more than the average size)
        else
        {
            print OUTPUT_FILE $rec || die "record:\n$rec\nfailed adding to file
            - aborting.\n";
            $l_output += $l_rec;
            start_new_file();
        }
        $ifile++;
    }
    close OUTPUT_FILE;
    print "Ended - split into $ifile output files.\n";
}
```

```
sub start_new_file {
    close OUTPUT_FILE;

    # recalc average size
    $size -= $l_output;
    $num_of_files = $size/$max_size + 1;
    $avg_size = $size / $num_of_files;
    if ( $avg_size == 0 )
    {
        $avg_size = $max_size;
    }

    $l_output++;
    open (OUTPUT_FILE, ">") . $outpref . $l_output ||
        die "couldn't open output file $outpref$l_output - aborting.\n";
    $l_output = 0;
}
```

```
#!/usr/local/bin/perl
use Getopt::Primargv;

# This script gets a file with the task lists in the format:
# <computer_name> <directory>
# and creates the command needed for the Guided merge of the files
# of the type wanted.
# in no file_prefix is given, we will assume that the prefix of the
# file is the same as the file_type.
#
Primargv("task_list=Task_list %s!file with lists of tasks to be merged", \%task_
list,
"contig_list= %s!file with list of all contigs", \%contig_list,
"output_dir= %s!output directory of files", \%output_dir,
"file_type= emb1 %s!file type (for structuredfile)", \%file_type,
"file_prefix= %s!file's prefix name in each directory", \%prefix);

# Get input parameters
if ( $prefix eq "" ) {
    $prefix = $file_type;
}

if ( $contig_list eq "" || $output_dir eq "" ) {
    die "Missing output directory or contig list!\n";
}

open (TASK_LIST, $task_list) || die "Couldn't open input file $task_list\n";

# prepare the command

$command = "GuidedMerge -file_type $(contig_list):list ";

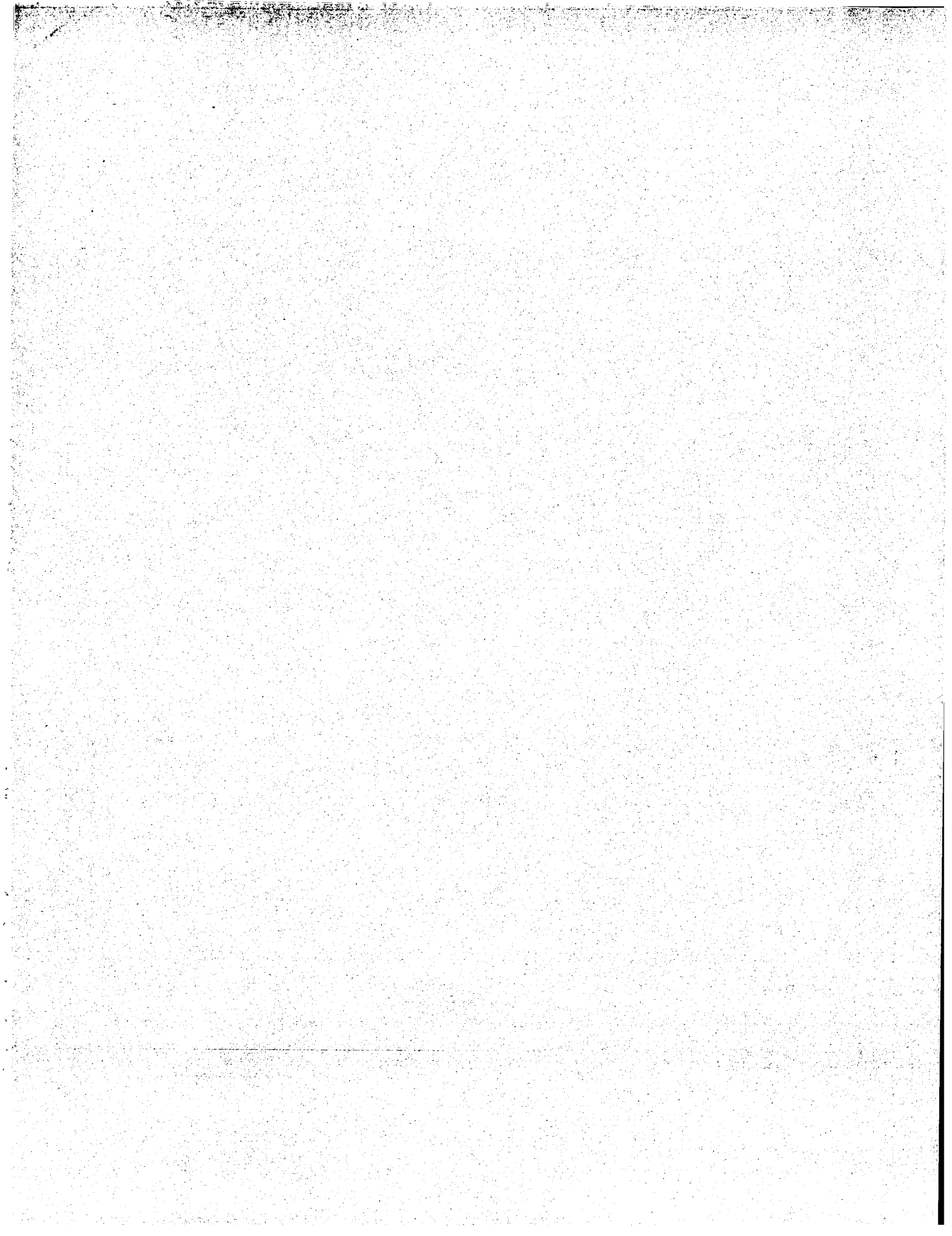
LOOP:
while (<TASK_LIST>) {
    if ( /^#/ ) {
        next LOOP;
    }
    (undef, $directory) = split;
    if ( ! $directory ) {
        die "Illegal task list line $_\n";
    }
    $command .= "$(directory)/$(prefix)* *";
}

$command .= ">$(output_dir)/$(prefix).merge 2>$(output_dir)/$(prefix).merge.errs";

# Execute the command

print "Going to execute the command:\n$(command)\n";
'$command';

if ( $? != 0 ) {
    print "The merge failed - check it!\n";
}
```

Using for Adam Sartel Sun Aug 9 10:45:44 1998

```
#!/usr/local/bin/perl

# check command line syntax
if(@ARGV != 2){
    die "usage is : sort_clusters <in_file> <out_file>";
}

# get query
$in=shift(@ARGV);
$out=shift(@ARGV);

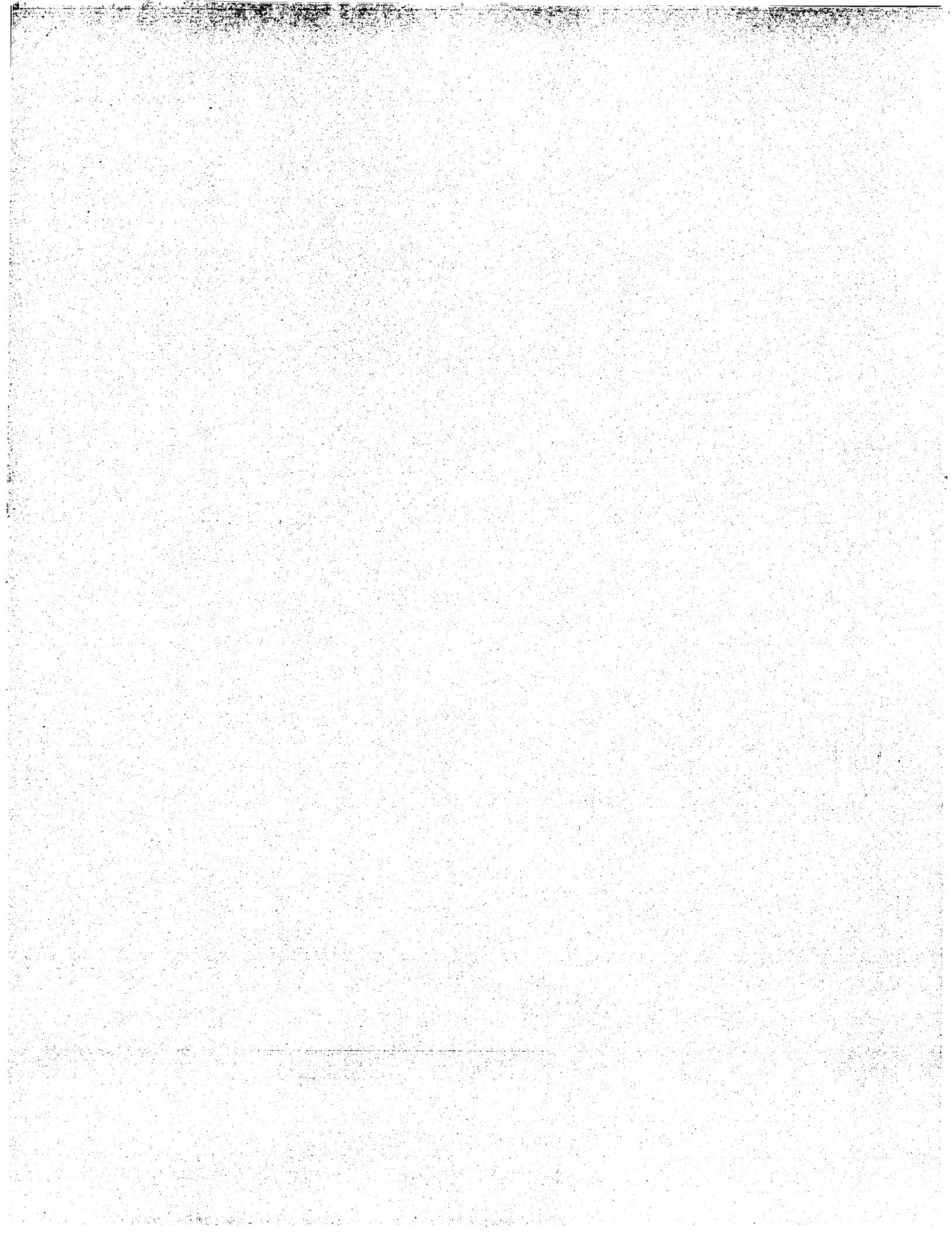
open (OUT, ">$out") || die "CRITICAL ERROR: can't open output file\n";
open (IN, $in) || die "CRITICAL ERROR: file $in not found\n";
$/ = "\n";
$i = 1;
while (<IN>) {
    s/^>///;
    ($name) = />(\S+)\s/;
    ($cluster) = /\#CU\s+CC.....(\S+)/;
    ($contig) = /\#CN\s+CC.....(\S+)/;
    s/>$/;
    $cluster{$name} = $cluster;
    $contig{$name} = $contig;
    $data{$name} = $_;
}
close (IN);

foreach $seq (sort by_cu_and_cn keys %cluster) {
    print OUT "$data{$seq}";
}

sub inv {a <=> b};

sub by_cu_and_cn {
    $x = ($cluster{$a} cmp $cluster{$b});
    if (!$x){
        $x = ($contig{$a} cmp $contig{$b});
    }
    return $x;
}

sub by_cluster { $cluster{$a} cmp $cluster{$b} };
```



```
#!/usr/local/bin/perl

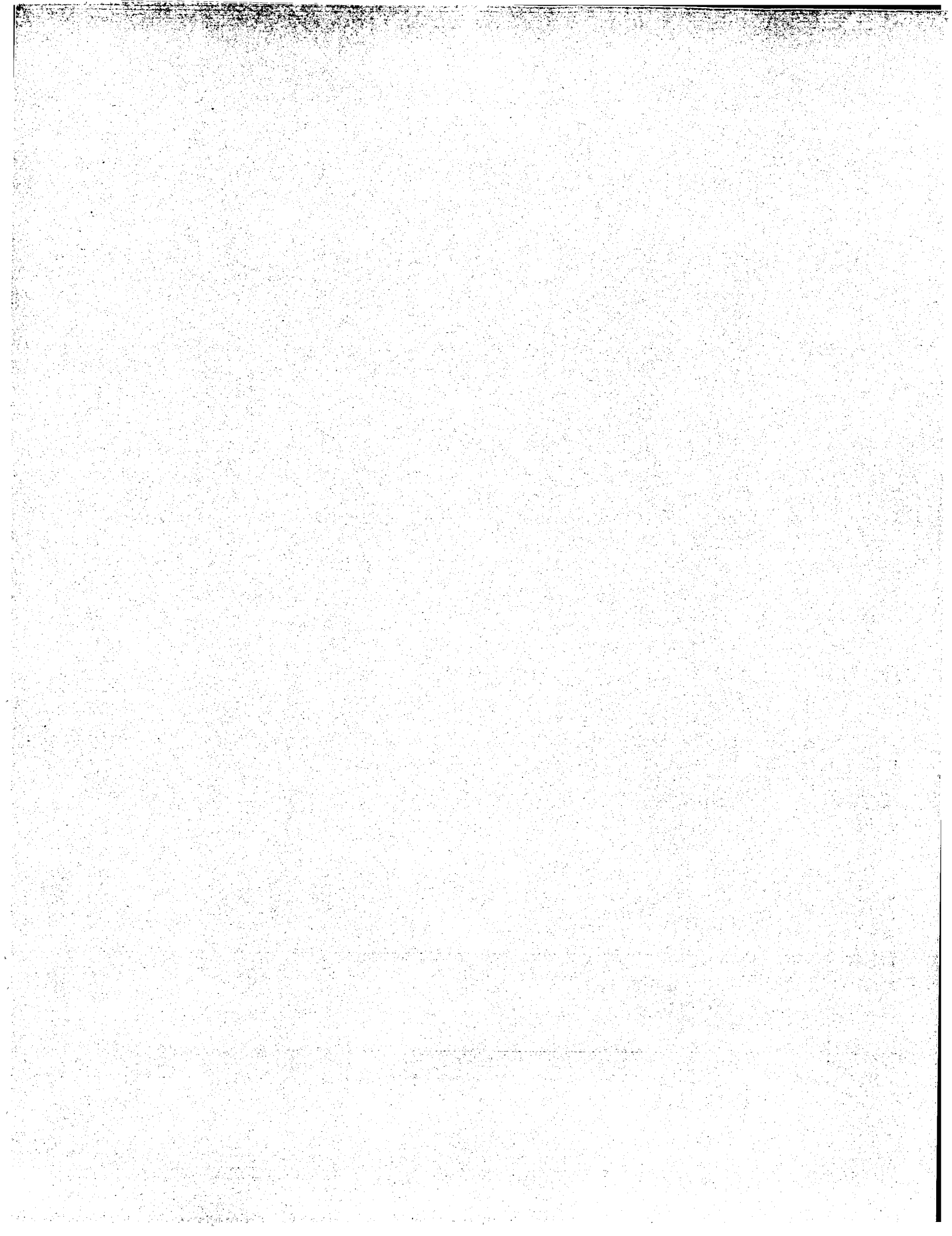
# check command line syntax
if(@ARGV != 3){
    die "usage is : sort_pairs.pl <list_file> <pairs_file> <out_file>";
}

# get query
$list=shift(@ARGV);
$name=shift(@ARGV);
$out=shift(@ARGV);

open (LIST, $list) || die "CRITICAL ERROR: file $list not found\n";
while (<LIST>) {
    ($name, $contig, $cluster) = /\s*(\S+)\s+(\S+)\s+(\S+)/;
    $cluster{$name} = $cluster;
    $contig{$name} = $contig;
}
close (LIST);

open (OUT, ">tmp") || die "CRITICAL ERROR: can't open output file\n";
open (IN, $in) || die "CRITICAL ERROR: file $in not found\n";
while (<IN>) {
    ($name1, $name2) = /\s*(\S+)\s+(\S+)/;
    $cluster = $cluster{$name1};
    $contig = $contig{$name1};
    print "Mismatching cluster $name1 $name2\n"
        if ($cluster != $cluster{$name2});
    print OUT "$cluster $contig $_";
}
close (IN);
close (OUT);

system "sort -k 1b tmp > $out";
system "/bin/rm tmp";
```



```
# OO routines to work with records in structured files. This file
# provides a virtual base-class for specific classes which deal with
# files.
```

```
# ariels 23/12/97
```

```
package StructuredFile;
```

```
use strict qw(refs vars);
```

```
use FileHandle;
```

```
use Carp;
```

```
# Constructor -- try to load appropriate package, then defer to it
sub new (
    no strict 'refs';
    my ($class, %args) = @_;
    my $constructor_name;
```

```
croak "Required TYPE field missing" unless $args{'type'};
require "$class/$args{'type'}.pm";
```

```
$class = "StructuredFile::$args{'type'}";
return &{"$(class)::new"}($class, %args);
```

```
# (should be returned properly blessed)
```

```
)
```

```
# Handy private function -- open underlying file
```

```
sub _open_file {
```

```
    my (%args) = @_;
```

```
    my $ret;
```

```
    unless ($ret{'fh'}) = new FileHandle $args{'name'}) {
```

```
        croak "Couldn't open $args{'name'}";
```

```
    }
```

```
    return \%ret;
```

```
}
```

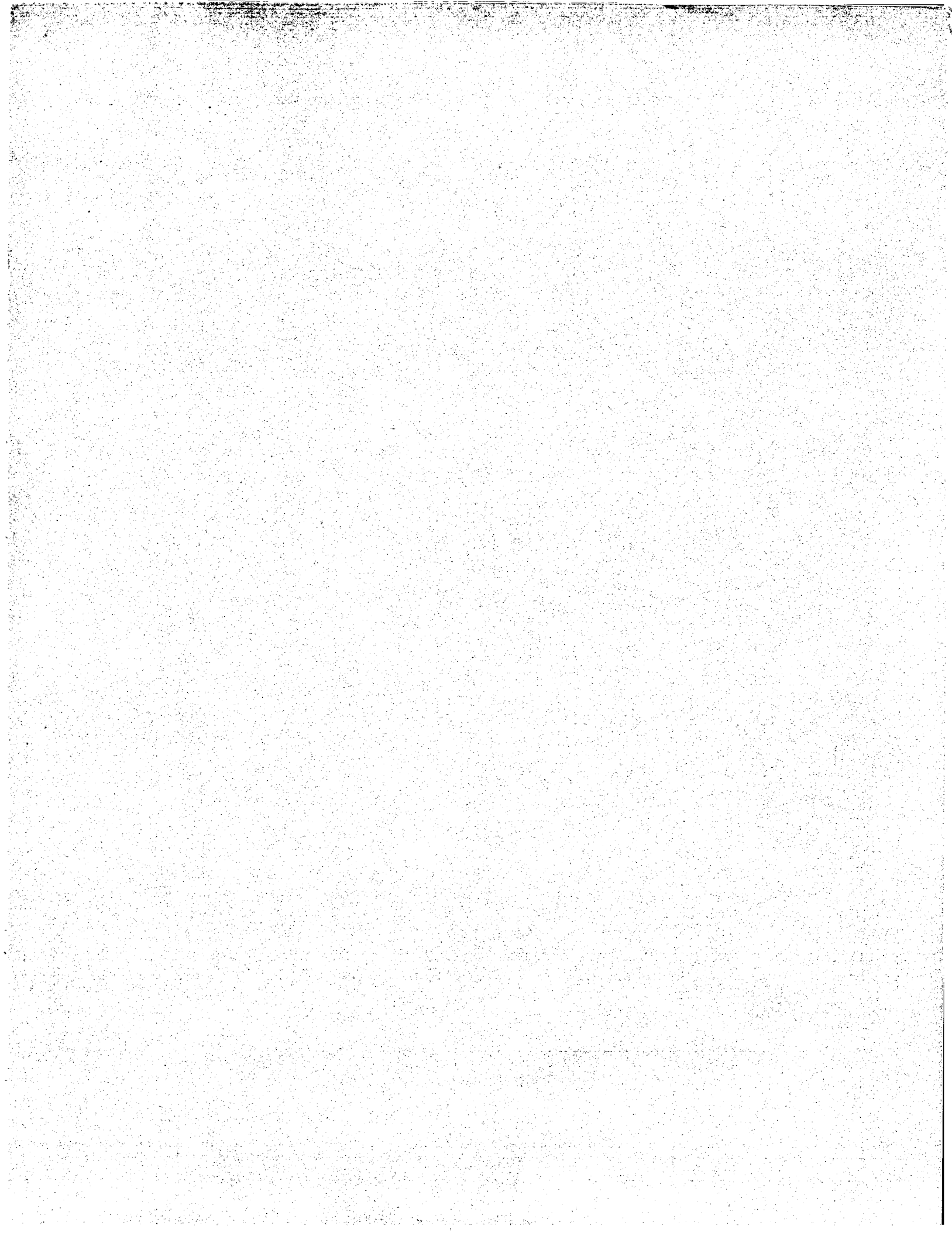
```
sub eof {
```

```
    my ($self) = @_;
```

```
    return eof $self->{'fh'};
```

```
}
```

```
1;
```



A-369

Listing for Adam Sartiel Sun Aug 9 10:49:50 1998

Listing for Adam Sartiel Sun Aug 9 10:49:50 1998

```
# Assembly standard-output structured file reader
#
# ariels 29/12/97
package StructuredFile::ass_out;

@ISA = qw(StructuredFile);

use strict qw(refs vars);

use Carp;
use FileHandle;

# Constructor
sub new {
    my ($class, %args) = @_;
    my $self;

    $self = StructuredFile::open_file(%args);

    # Skip past initial argument list
    do {
        $self->('line') = $self->('fh')->getline;
    } while ($self->('line') !~ /\s/);

    return bless $self, $class;
}

sub _skip_blanks {
    my ($self) = @_;

    $self->('line') = $self->('fh')->getline
        while (defined($self->('line')) && $self->('line') =~ /\s$/);
}

sub get_record {
    my ($self) = @_;
    my ($contig, $rec);

    $self->('line') || return undef;

    $self->('line') =~ /\s/ || croak "No header found in \"$self->('line')\"";
    $rec .= $self->('line');
    $self->('line') = $self->('fh')->getline;

    $contig = $1 if ($self->('line') =~ /\s*\s*begin\s*([^\s]*)\s/);
    unless (defined($contig)) {croak "Bad format \"$self->('line')\"";}

    while ($self->('line') =~ /\s*$/) {
        $rec .= $self->('line');
        $self->('line') = $self->('fh')->getline;
    }

    while (defined($self->('line')) && $self->('line') !~ /\s*\s*begin\s*([^\s]*)\s/ &&
        $self->('line') !~ /\s*\s*end\s*([^\s]*)\s/) {
        $rec .= $self->('line');
    }

    if ($self->('line') =~ /\s*\s*report/) {
        $self->('line') = $self->('fh')->getline;
    }

    if ($self->('line') =~ /\s*\s*Can't open output file/) {
        # Hack because these lines don't have EOL, and mark the end of the
        # useful portion of the file anyway
        $self->('line') = undef;
        return undef;
    }

    $self->_skip_blanks;

    $rec .= "\n";
    # Append a blank line

    return ($rec, $contig);
}

sub eof {
    my ($self) = @_;

    return (! defined($self->('line')));
}

1;
```

ass_out.pm

ass_out.pm

Using for Adam Sartiell Sun Aug 9 10:49:51 1998

```
# Assembly cluster-view output structured file reader
#
# ariels 23/12/97

package StructuredFile::cv;

@ISA = ("StructuredFile");

use strict qw(refs vars);

use Carp;
use FileHandle;

# Constructor
sub new {
    my ($class, %args) = @_;
    my $self;
    my $line;

    $self->{'fh'} = new FileHandle;
    $self->{'fh'}->open($args{'name'}) ||
        croak "Couldn't open $args{'name'}";

    # chomp($line = $self->{'fh'}->getline);
    # $line =~ /^VER assembly 1./i || croak "Bad version line $line";

    return bless \%self, $class;
}

sub get_record {
    my ($self) = @_;
    my ($cluster, $contig);
    my ($rec, $line);

    return undef if $self->{'fh'}->eof;

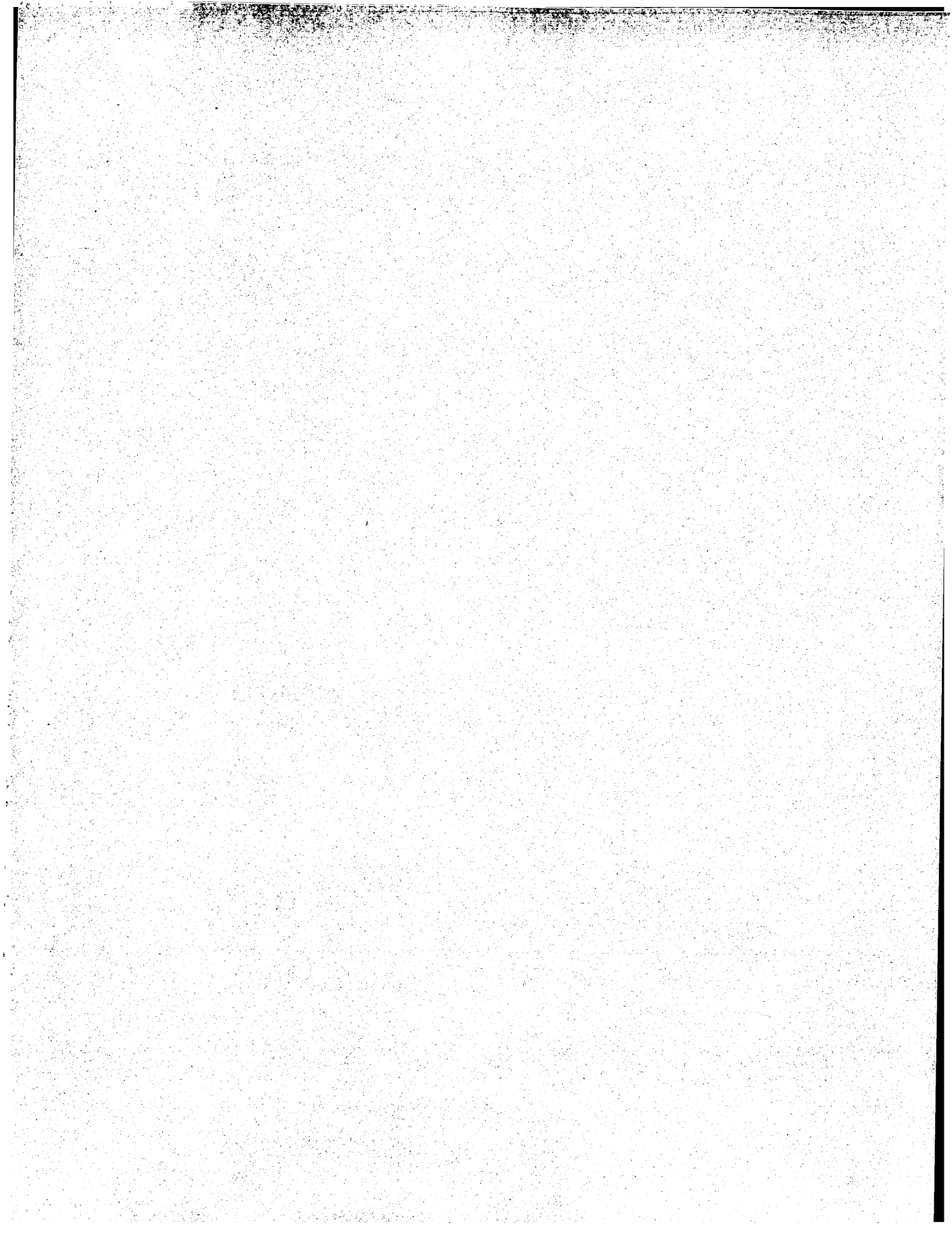
    $rec = "";
    do {
        ($line = $self->{'fh'}->getline) or return undef;
        $contig = $1 if ($line =~ m/^ID +([^\n]*)/);
        $cluster = $1 if ($line =~ m/^CLUSTER +([^\n]*)/);
        $rec .= $line;
    } while ($line !~ m/^//);

    croak "Bad format \"$rec\" unless ($contig && $cluster);

    return ($rec, $cluster.$contig);
}

sub eof {
    my ($self) = @_;
    return ($self->{'fh'}->eof);
}

1;
```



Listing for Adam Sarteil Sun Aug 9 10:49:51 1998

```
# Assembly EMBL output structured file reader
# # ariels 28/12/97
package StructuredFile::embl;
@ISA = ("StructuredFile");
use strict qw(refs vars);

use Carp;
use FileHandle;

# Constructor
sub new {
    my ($class, %args) = @_;
    my %self;

    $self{'fh'} = new FileHandle;
    $self{'fh'}->open($args{'name'}) ||
        croak "couldn't open $args{'name'}";
    return bless \%self, $class;
}

sub get_record {
    my ($self) = @_;
    my ($contig); # Currently no cluster
    my ($rec, $line);

    return undef if $self->{'fh'}->eof;
    $rec = "";
    do {
        $line = $self->{'fh'}->getline;
        $contig = $1 if ($line =~ m/^ID +([^\n]*)/);
        $rec = $line;
    } while ($line !~ m/^\/$/);
    croak "Bad format \"$rec\" unless $contig;
    return ($rec, $contig);
}

sub eof {
    my ($self) = @_;
    return ($self->{'fh'}->eof);
}

1;
```



```

# FastA structured file reader
#
# avir 28/5/1998
package StructuredFile::fasta;
$ISA = qw(StructuredFile);

use strict qw(refs vars);

use Carp;
use FileHandle;

# Constructor
sub new {
    my ($class, %args) = @_;
    my $self;

    $self = StructuredFile::_open_file(%args);
    $self->{'line'} = $self->{'fh'}->getline;
    return bless $self, $class;
}

sub get_record {
    my ($self) = @_;
    my ($cluster, $contig);
    my $rec;

    $self->{'line'} || return undef;
    if ($self->{'line'} =~ /^>/) {
        $cluster = $1 if $self->{'line'} =~ /^>([^\n]*)/;
        $contig = $1 if $self->{'line'} =~ /^>([^\n]*)/;
    }
    unless ($cluster && $contig) {
        croak "Bad format \"$self->{'line'}\"";
    }
    $rec = $self->{'line'};
    while (defined($self->{'line'}) = $self->{'fh'}->getline) &&
        $self->{'line'} !~ /^>/ {
        $rec .= $self->{'line'};
    }
    return ($rec, "$cluster.$contig");
}

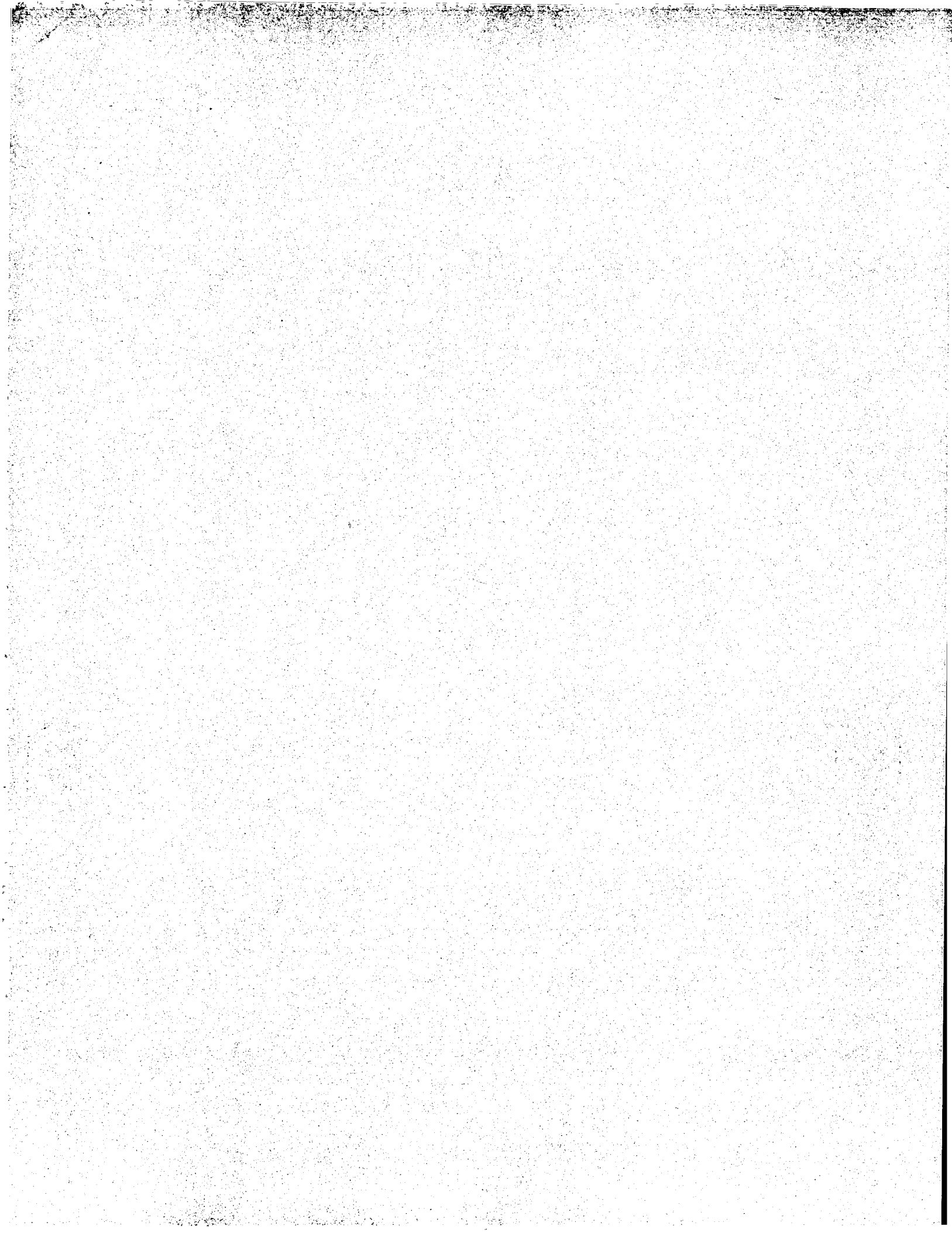
sub eof {
    my ($self) = @_;
    return (! defined($self->{'line'}));
}

```

```

1;

```



```
# Assembly EMBL output structured file reader
#
# ariels 28/12/97

package StructuredFile::graph;

@ISA = ("StructuredFile");

use strict;

use Carp;
use FileHandle;

# Constructor
sub new (
    my ($class, %args) = @_ ;
    my $self;

    $self = StructuredFile::_open_file(%args);
    return bless $self, $class;
)

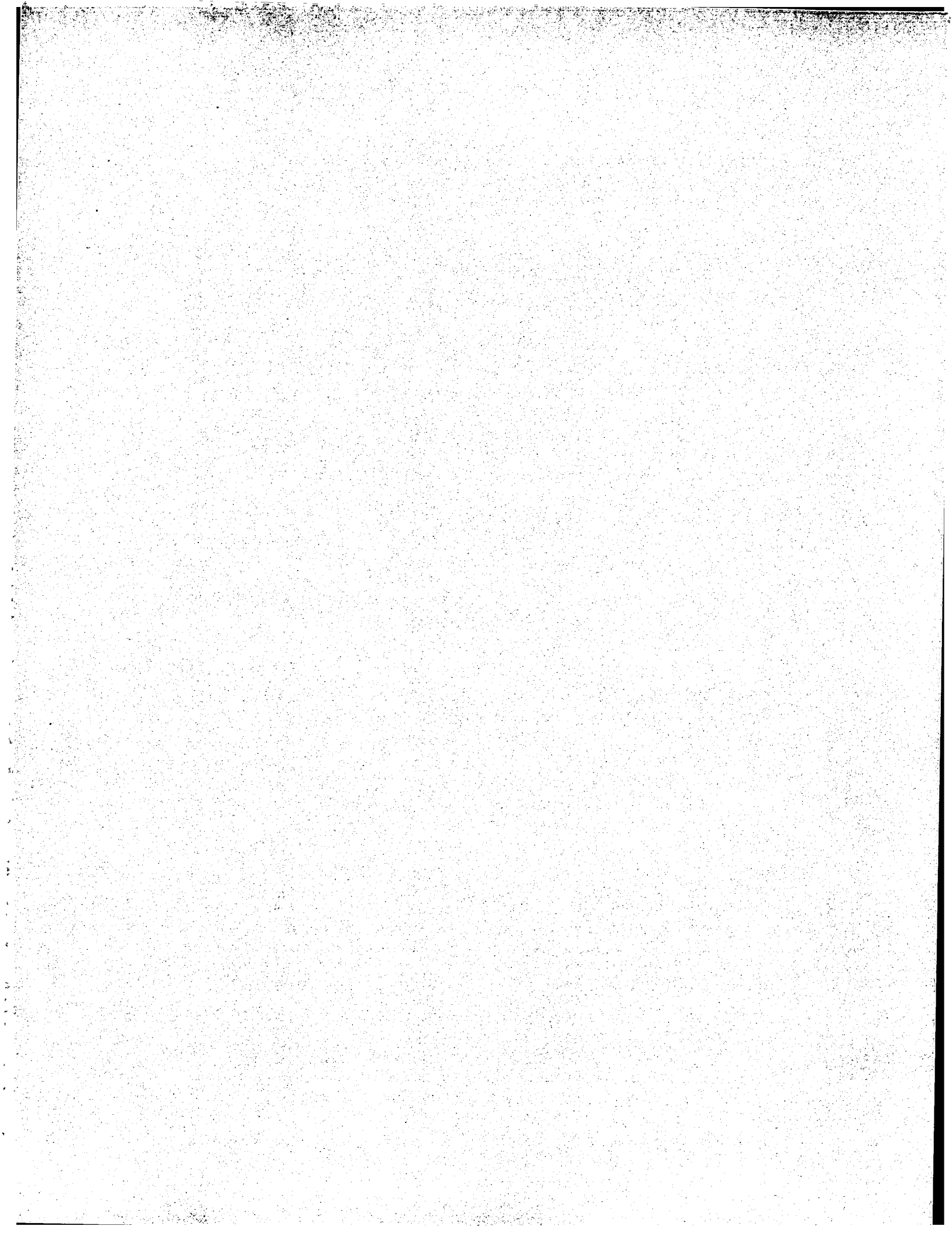
sub get_record {
    my ($self) = @_;
    my ($contig,$cluster);
    my ($rec, $line);

    return undef if $self->{'fh'}->eof;

    $rec = "";
    do {
        $line = $self->{'fh'}->getline;
        if ($rec eq "") { # First line
            croak "Missing CONTIG cluster.contig line, got \"$line\"";
        } unless ($cluster, $contig) =
            $line =~ /^CONTIG (\w+)\.(\w+)/;
    }
    $rec .= $line;
    } while ($line !~ m/^ENDCONTIG$/);
    return ($rec, "$cluster.$contig");
}

sub eof {
    my ($self) = @_;
    return ($self->{'fh'}->eof);
}

1;
```

Listing for Adam Santiel Sun Aug 9 10:49:51 1998

```
# Assembly keyword output structured file reader
#
# ariels 28/12/97
package StructuredFile::keyword;
@ISA = ("StructuredFile");
use strict qw(refs vars);

use Carp;
use FileHandle;

# Constructor
sub new {
    my ($class, %args) = @_;
    my %self;

    $self{'fh'} = new FileHandle;
    $self{'fh'}->open($args{'name'}) ||
        croak "Couldn't open $args{'name'}";
    return bless \%self, $class;
}

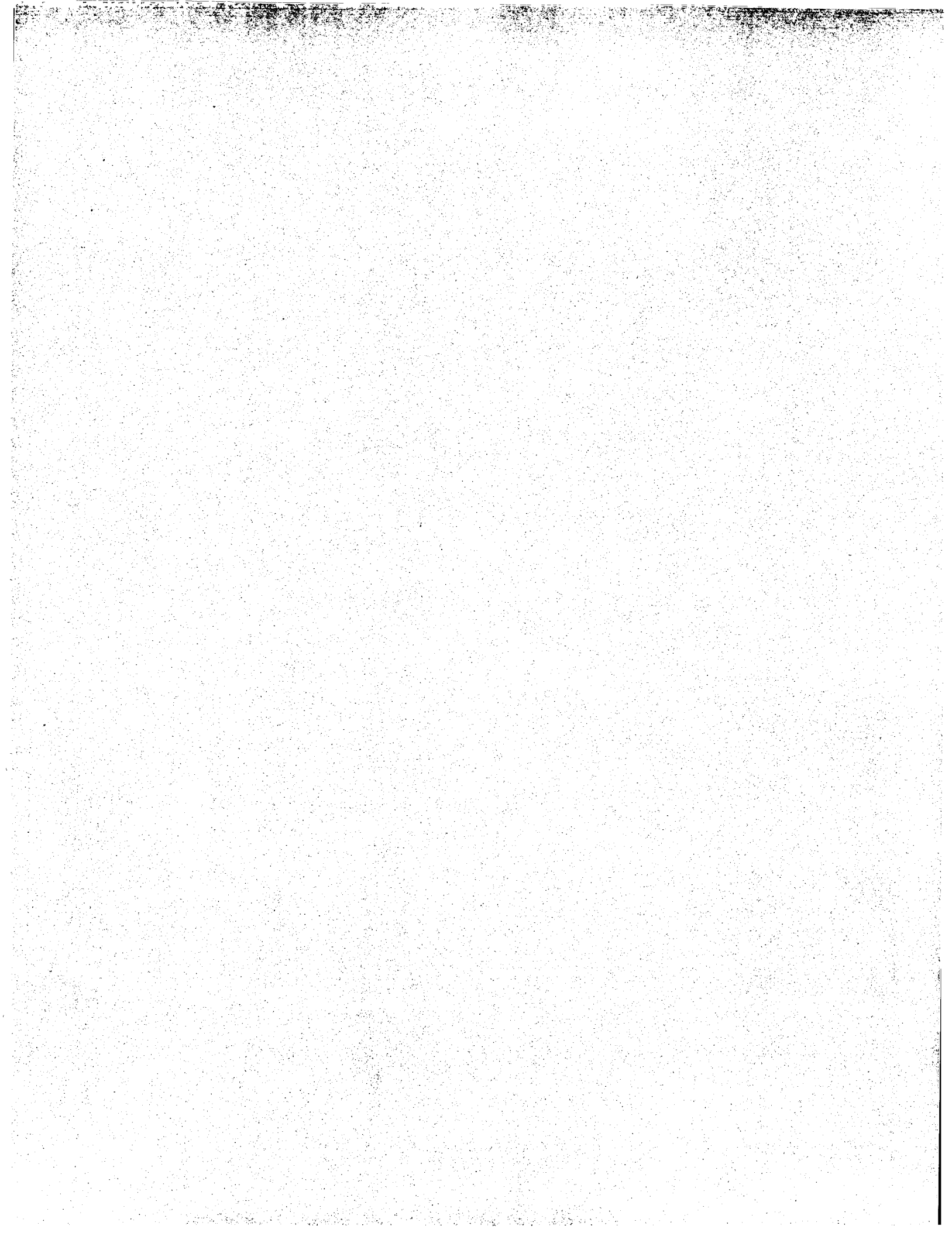
sub get_record {
    my ($self) = @_;
    my ($contig, $line);

    return undef if $self->{'fh'}->eof;
    $line = $self->{'fh'}->getline;
    $contig = $1 if $line =~ /^(\w*) /;

    croak "Bad format \"$line\" unless $contig;
    return ($line, $contig);
}

sub eof {
    my ($self) = @_;
    return ($self->{'fh'}->eof);
}
```

1;



Sun Aug 9 10:49:52 1998

Listing for Adam Santiel

```

# "List" -- "CLUSTER.CONTIG gibberish" structured file reader
#
# ariels 4/3/98

package StructuredFile::list;

@ISA = qw(StructuredFile);

use strict;

use Carp;
use FileHandle;

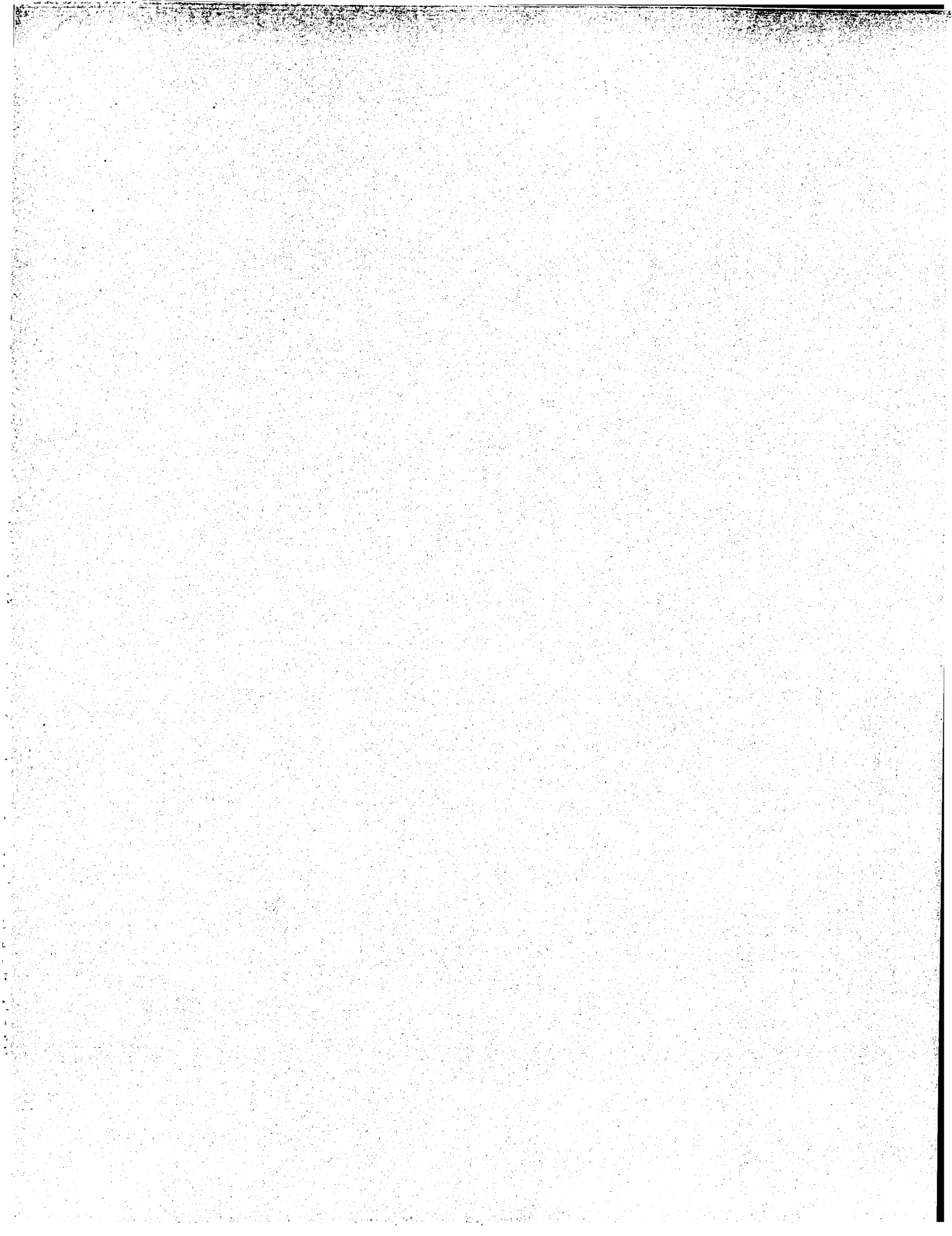
# Constructor
sub new (
    my ($class, %args) = @_ ;
    my $self;

    $self = StructuredFile::_open_file(%args);
    return bless $self, $class;
)

sub get_record (
    my $self = shift;
    my ($cluster, $contig);
    my $line = $self->{'fh'}->getline;
    return undef unless defined($line);

    if ($line =~ /^(\\w+)\\. (\\w+).*$/) {
        $cluster = $1;
        $contig = $2;
    }
    else {
        croak "Bad format \"$line\"";
    }
    return ($line, "$cluster.$contig");
)

```



Listing for Adam Sartei

Sun/Aug 9 10:49:52 1998

```

# Rich-Pasta structured file reader
#
# ariels 23/12/97
package StructuredFile::rf;
@ISA = qw(StructuredFile);

use strict qw(refs vars);

use Carp;
use FileHandle;

# Constructor
sub new {
    my ($class, %args) = @_;
    my $self;
    $self = StructuredFile::_open_file(%args);
    $self->('line') = $self->('fh')->getline;
    return bless $self, $class;
}

sub get_record {
    my ($self) = @_;
    my ($cluster, $contig);
    my $rec;
    $self->('line') || return undef;
    if ($self->('line') =~ /^>/) {
        $cluster = $1 if $self->('line') =~ /\#CU +([^\n]*)/;
        $contig = $1 if $self->('line') =~ /\#CN +([^\n]*)/;
    }
    unless ($cluster && $contig) {
        croak "Bad format \"$self->('line')\"";
    }
    $rec = $self->('line');
    while (defined($self->('line') = $self->('fh')->getline) &&
        ($self->('line') !~ /^>/ ||
         $self->('line') =~ /\#CN \Q$contig\E /)) {
        $rec .= $self->('line');
    }
    return ($rec, *$cluster.$contig);
}

sub eof {
    my ($self) = @_;
    return (! defined($self->('line')));
}

```

rf.pm

Listing for Adam Sartei

Sun/Aug 9 10:49:52 1998

1;

rf.pm

A-376

Using for Adam Santel Sun Aug 9 10:49:52 1998

```
# Transcript-database structured file reader
#
# ariels 28/12/97
package StructuredFile::transc_db;

@ISA = qw(StructuredFile);

use strict qw(refs vars);

use Carp;
use FileHandle;

# Constructor
sub new {
    my ($class, %args) = @_;
    my $self;

    $self->{'fh'} = new FileHandle;
    $self->{'fh'}->open($args{'name'}) ||
        croak "Couldn't open $args{'name'}";

    $self->{'line'} = $self->{'fh'}->getline;

    return bless \$self, $class;
}

sub get_record {
    my ($self) = @_;
    my ($contig, $rec);

    $self->{'line'} || return undef;

    $contig = $1 if $self->{'line'} =~ /^>\w*_\w*\d* /;
    croak "Bad format \"$self->{'line'}\" unless $contig;

    $rec = $self->{'line'};
    while (defined($self->{'line'}) = $self->{'fh'}->getline) &&
        ($self->{'line'}) !~ /^>/ ||
        $self->{'line'} =~ /^>\w*_\w*\d* /) {
        $rec .= $self->{'line'};
    }

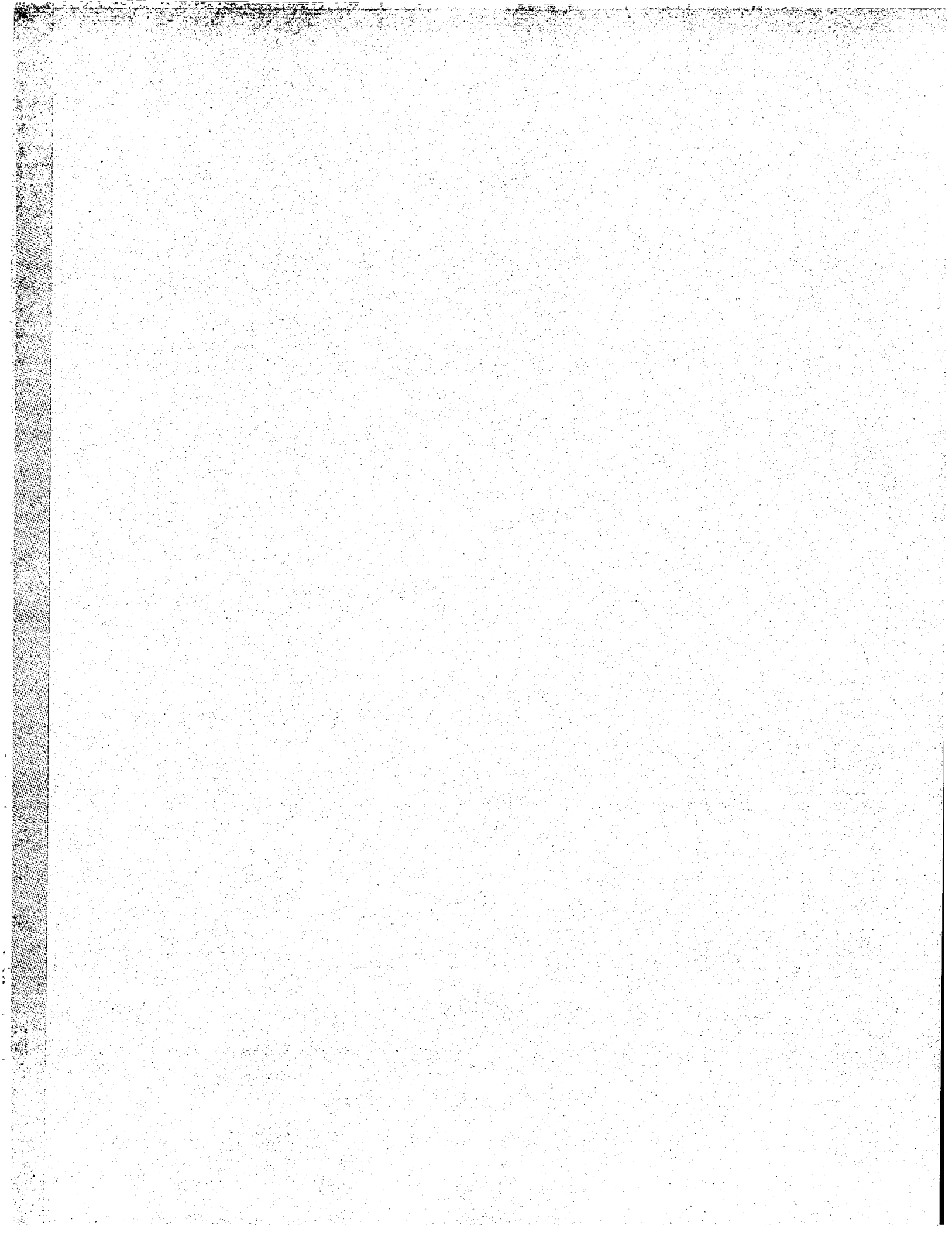
    return ($rec, $contig);
}

sub eof {
    my ($self) = @_;

    return (! defined($self->{'line'}));
}

1;
```

transc_db.pm



```
# GetOpt::PmArgv.pm -- Universal options parsing
```

```
package Getopt::PmArgv;
```

```
# RCS Status      : Avi Rosenberg
# Created On      : Tue Mar 3 16:30:00 1998
# Last Modified By: Avi Rosenberg
# Update Count    :
# Status          :
```

```
=head1 NAME
```

```
PmArgv- extended processing of command line options
```

```
=head1 SYNOPSIS
```

```
use Getopt::PmArgv;
$result = PmArgv (...option-descriptions...);
```

```
=head1 DESCRIPTION
```

The Getopt::PmArgv module implements an extended getopt function called PmArgv(). This function allows giving options on the command line and binding the appropriate values to variables.

Command line options can be used to set values. These values can be specified in one of two ways:

```
-size 24
size=24
```

PmArgv is called with a list of option-descriptions, each of which consists of two elements: the option specifier and the option linkage. The option specifier can in one of the following formats:

1. "<field> = <def.value> <format> <*> !<remark>"
If * appears, it means that the field is an array, and more than one value can be assigned to it. Assigning an array is done by specifying the field on the command line the number of times needed.

def.value and remark are optional.

The legal formats are:

```
%d - integer value.
%f - float (real) value.
%s - string value.
```

2. "-<flag> <-> !<remark>"
The default value for the flag is 1 unless the sign '-' is used. remark is optional.

The option linkage is a reference to a variable that will be set when the option is used.

For example, the following call to PmArgv:

```
PmArgv("size = %d !size of offset", \%offset);
```

will accept a command line option "size" that must have an integer value. With a command line of "-size 24" this will cause the variable \%offset to get the value 24.

The command line options are taken from array @ARGV. Upon completion

of PmArgv, @ARGV will contain the rest (i.e. the non-options) of the command line.

PmArgv will print out the value of the fields that were given on the command line, or had default values, to STDERR.

```
=head1 EXAMPLES
```

If the option specifier is "one = %d" (i.e. takes an integer argument), then the following situations are handled:

```
-one -2      -> $my_var = -2
one= 3       -> $my_var = 3
```

Example of using variable references:

```
$ret = PmArgv ('foo= %s!', \%foo, 'bar= %s', \%bar);
```

With command line options "-foo blech -bar 24 -ar xx -ar yy" this will result in:

```
$foo = 'blech'
$bar = 24
@ar = ('xx', 'yy')
```

```
=head1 BUGS and FEATURES
```

This package uses the Evap package for the evaluation of the command line parameters. If you don't have the Evap package, it is available via anonymous FTP from ftp.lehigh.edu (128.180.1.4) in the directory pub/evap/evap-2.x.

Since the command line interface of PmArgv and Evap are a little different, PmArgv modified the ARGV global variable, and this may affect some usages.

```
=back
```

```
=cut
```

```
##### Copyright #####
```

```
# This program is Copyright 1990, 1998 by Avi Rosenberg
# This program is free software; you can redistribute it and/or
# modify it under the terms of the GNU General Public License
# as published by the Free Software Foundation; either version 2
# of the License, or (at your option) any later version.
```

```
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
# GNU General Public License for more details.
# The author will NOT BE HELD RESPONSIBLE for any error resulting
# of using this program. It is NOT RECOMMENDED to use this program
# in ANY LIFE CRITICAL usage.
```

```
# If you do not have a copy of the GNU General Public License write to
# the Free Software Foundation, Inc., 675 Mass Ave, Cambridge,
# MA 02139, USA.
```

```
##### Module Preamble #####
use strict;
use Getopt::Evap;

BEGIN {
    require 5.003;
    use Exporter ();
    use vars qw(@ISA @EXPORT);
    @ISA = qw(Exporter);
    @EXPORT = qw(PrmArgv);
}

##### Local Variables #####
my @PDT;
my @MM;

##### Subroutines #####

sub PrmArgv {
    my @optionlist = @_;
    my @new_argv;

    my $current_arg;
    my $new_arg1;
    my $new_arg2;

    my %ref_hash;
    my %flag_hash;
    my %array_hash;

    my $field_name;

    my $default_value;
    my $given_format;
    my $is_array;
    my $remark;

    my $negate_flag;

    my $ref_var;
    my $eval_var;

    my $pdt_entry;

    my %output_hash;

    # create the PDT, MM, ref_hash, flag_hash, array_hash

    @PDT = ("PDT $0");
    @MM = (" $0", "?");

    while (@optionlist > 1)
    {
        $current_arg = shift(@optionlist);

        # local copy of the option descriptions
        # modified copy of ARGV - switching
        # "<field=value" to "-field", "value"
        # The current arg in ARGV

        # hash of variable references
        # hash of flags (0 regular, 1 negated)
        # hash of array variables

        # values of field specifiers

        # value for flag specifier

        # The current output reference variable
        # The value of the current output

        # The current entry in PDT

        # The output hash for Evap

        # create the PDT, MM, ref_hash, flag_hash, array_hash

        @PDT = ("PDT $0");
        @MM = (" $0", "?");

        while (@optionlist > 1)
        {
            $current_arg = shift(@optionlist);
```

```
# Check if specifier is of a regular field
#
if ( ($field_name, $default_value, $given_format, $is_array, $remark) =
    ( $current_arg =~
        /\^(w+)(s+\.*)?\s+&{([dfs])}\s+(\.*)?s!(.)*?$/s ) )
{
    # Update MM entry
    #
    push @MM, " " . $field_name, $remark;

    # Create PDT entry and update PDT
    #
    $pdt_entry = $field_name . " " . $field_name . " "; # field name a
    nd abbr.

    if ( $is_array )
    {
        $array_hash{$field_name} = 1;
        $pdt_entry .= "list of ";
    }

    if ($given_format eq "d")
    {
        $pdt_entry .= "integer";
    }
    elsif ($given_format eq "f")
    {
        $pdt_entry .= "real";
    }
    elsif ($given_format eq "s")
    {
        $pdt_entry .= "string";
    }
    else
    {
        print STDERR "Illegal format specifier \"$given_format\".\n";
        exit(1);
    }

    if ( defined $default_value )
    {
        if ( $is_array )
        {
            $pdt_entry .= " = ( " . $default_value . " )";
        }
        else
        {
            $pdt_entry .= " = " . $default_value;
        }
    }

    push @PDT, $pdt_entry;
}
}
```



```
# # print out the parameters read, and put them in the variables
# #
foreach $field_name (keys %output_hash)
(
    $eval_var = $output_hash{$field_name};
    $ref_var = $ref_hash{$field_name};
    if ( exists $flag_hash{$field_name} )
    {
        print STDERR "$field_name\n";
        # flag appeared - change the value of the variable
        #
        $$ref_var = 1 - $$ref_var;
    }
    else
    {
        print STDERR $field_name, " = ";
        if ( exists $array_hash{$field_name} )
        {
            @$ref_var = @$eval_var;
            print STDERR join(" ", @$ref_var);
        }
        else
        {
            $$ref_var = $eval_var;
            print STDERR $$ref_var;
        }
        print STDERR "\n";
    }
} # end PmArgv
1;
```

```

package Getopt::EvaP;

# EvaP.pm - Evaluate Parameters 2.3 for Perl (the getopt et.al. replacement)
# lusol@lehigh.EDU, 94/10/28
#
# Made to conform, as much as possible, to the C function evap. The C, Perl
# and Tcl versions of evap are patterned after the Control Data procedure
# CLP$EVALUATE_PARAMETERS for the NOS/VE operating system, although none
# approach the richness of CDC's implementation.
#
# Availability is via anonymous FTP from ftp.lehigh.EDU (128.180.1.4) in the
# directory pub/evap/evap-2.x.
#
# Stephen O. Lidie, Lehigh University Computing Center.
#
# Copyright (C) 1993 - 1997 by Stephen O. Lidie. All rights reserved.
#
# This program is free software; you can redistribute it and/or modify it under
# the same terms as Perl itself.
#
# For related information see the evap/C header file evap.h. Complete
# help can be found in the man pages evap(2), evap.c(2), EvaP.pm(2),
# evap.tcl(2) and evap_pac(2).
#
=head1 NAME

Getopt::EvaP() - evaluate Perl command line parameters.

=head1 SYNOPSIS

use vars qw($PDT $MM $OPT);
use Getopt::EvaP;

EvaP $PDT, $MM, $OPT;

=head1 DESCRIPTION

B<@PDT>
is the Parameter Description Table, which is a reference to a list of
strings describing the command line parameters, aliases,
types and default values.
B<@MM>
is the Message Module, which is also a reference to a list of strings
describing the command and it's parameters.
B<@OPT>
is an optional hash reference where Evaluate Parameters should place its
results. If specified, the historical behaviour of modifying the calling
routines' namespace by storing option values in B<@options>, B<@options> and
B<@opt> is disabled.

B<+ Introduction +>

Function Evaluate Parameters parses a Perl command line in a simple and
consistent manner, performs type checking of parameter values, and provides
the user with first-level help. Evaluate Parameters is also embeddable in
your application; refer to the B<evap_pac(2)> man page for complete details.
Evaluate Parameters handles command lines in the following format:

```

```

command [-parameters] [file_list]

where parameters and file_list are all optional. A typical example is the
C compiler:

cc -O -o chunk chunk.c

In this case there are two parameters and a file_list consisting of a
single file name for the cc command.

B<+ Parameter Description Table (PDT) Syntax +>

Here is the PDT syntax. Optional constructs are enclosed in [], and the
| character separates possible values in a list.

PDT (program_name, alias)
[parameter_name[, alias]: type [= {default_variable;} default_value]]
PDTEND [optional_file_list | required_file_list | no_file_list]

So, the simplest possible PDT would be:

PDT
PDTEND

This PDT would simply define a I<-help> switch for the command, but is rather
useless.

```

A typical PDT would look more like this:

```

PDT frog
  number, n: integer = 1
PDTEND no_file_list

```

This PDT, for command frog, defines a single parameter, number (or n), of type integer with a default value of 1. The PDTEND I<no_file_list> indicator indicates that no trailing file_list can appear on the command line. Of course, the I<-help> switch is defined automatically.

The I<default_variable> is an environment variable - see the section Usage Notes for complete details.

B<+ Usage Notes +>

Usage is similar to getopt/getopts/newgetopt: define a Parameter Description Table declaring a list of command line parameters, their aliases, types and default values. The command line parameter I<-help> (alias I<-h>) is automatically included by Evaluate Parameters. After the evaluation the values of the command line parameters are stored in variable names of the form B<\$opt_parameter>, except for lists which are returned as B<\$opt_parameter>, where I<parameter> is the full spelling of the command line parameter. NOTE: values are also returned in the hashes B<\$options> and B<\$opt>, with lists being passed as a reference to a list.

Of course, you can specify where you want Evaluate Parameters to return its results, in which case this historical feature of writing into your namespace

is disabled.

An optional PDT line can be included that tells Evaluate Parameters whether or not trailing file names can appear on the command line after all the parameters. It can read I<no_file_list>, I<optional_file_list> or I<required_file_list> and, if not specified, defaults to optional. Although placement is not important, this line is by convention the last line of the PDT declaration.

Additionally a Message Module is declared that describes the command and provides examples. Following the main help text an optional series of help text messages can be specified for individual command line parameters. In the following sample program all the parameters have this additional text which describes that parameter's type. The leading character is a dot in column one followed by the full spelling of the command line parameter. Use I<-full help> rather than I<-help> to see this supplemental information. This sample program illustrates the various types and how to use B<EvaP()>. The I<key> type is a special type that enumerates valid values for the command line parameter. The I<boolean> type may be specified as TRUE/FALSE, YES/NO, ON/OFF or 1/0. Parameters of type I<file> have ~ and \$HOME expanded, and default values I<stdin> and I<stdout> converted to '-' and '>', respectively. Of special note is the default value I<\$required>; when specified, Evaluate Parameters will ensure a value is specified for that command line parameter.

All lists except I<switch> may be I<list of>, like the I<tty> parameter below. A list parameter can be specified multiple times on the command line. NOTE: in general you should ALWAYS quote components of your lists, even if they're not type string, since Evaluate Parameters uses eval to parse them. Doing this prevents eval from evaluating expressions that it shouldn't, such as file name shortcuts like \$HOME, and backticked items like 'hostname'. Although the resulting PDT looks cluttered, Evaluate Parameters knows what to do and eliminates superfluous quotes appropriately.

Finally, you can specify a default value via an environment variable. If a command line parameter is not specified and there is a corresponding environment variable defined then Evaluate Parameters will use the value of the environment variable. Examine the I<command> parameter for the syntax. With this feature users can easily customize command parameters to their liking. Although the name of the environment variable can be whatever you choose, the following scheme is suggested for consistency and to avoid conflicts in names:

=over 4

=item *

Use all uppercase characters.

=item *

Begin the variable name with D_ to suggest a default variable.

=item *

Continue with the name of the command or its alias followed by an underscore.

=item *

Complete the variable name with the name of the parameter or its alias.

=back

So, for example, D_DISC1 DO would name a default variable for the display option (do) parameter of the display_command_information (disci) command. Works for MS-DOS and Unix.

Example:

```
#!/usr/local/bin/perl
use Getopt::EvaP;

@PDT = split /\n/, <<'end-of-PDT';
PDT sample
verbose, v: switch
command, c: string = D_SAMPLE_COMMAND, "ps -el"
scale_factor, sf: real = 1.2340896e-1
millisecond_update_interval, mui: integer = $required
ignore_output_file_column_one, iofco: boolean = TRUE
output, o: file = stdout
queue, q: key plotter, postscript, text, printer, keyend = printer
destination, d: application = 'hostname'
tty, t: list of name = ("/dev/console", "/dev/tty0", "/dev/tty1")
PPTEND optional_file_list
end-of-PDT

@MM = split /\n/, <<'end-of-MM';
sample
```

A sample program demonstrating typical Evaluate Parameters usage.

Examples:

```
sample
sample -usage_help
sample -help
sample -full_help
sample -mui 1234
```

.verbose

A switch type parameter emulates a typical standalone switch. If the switch is specified Evaluate Parameters returns a '1'.

.command

A string type parameter is just a list of characters, which must be quoted if it contains whitespace.

NOTE: for this parameter you can also create and initialize the environment variable D_SAMPLE_COMMAND to override the standard default value for this command line parameter. All types except switch may have a default environment variable for easy user customization.

.scale_factor

A real type parameter must be a real number that may contain a leading sign, a decimal point and an exponent. millisecond_update_interval

An integer type parameter must consist of all digits with an optional leading sign. NOTE: this parameter's default value is \$required, meaning that

Evaluate Parameters ensures that this parameter is specified and given a valid value. All types except switch may have a default value of \$required.

ignore_output_file_column_one
A boolean type parameter may be TRUE/YES/ON/1 or FALSE/NO/OFF/0, either upper or lower case. If TRUE, Evaluate Parameters returns a value of '1', else '0'.

.output
A file type parameter expects a filename. For Unix \$HOME and ~ are expanded. For Evap/Perl stdin and stdout are converted to '-' and '>-' so they can be used in a Perl open() function.

.queue
A key type parameter enumerates valid values. Only the specified keywords can be entered on the command line.

.destination
An application type parameter is not type-checked in any - the treatment of this type of parameter is application specific. NOTE: this parameter's default value is enclosed in grave accents (or "backticks"). Evaluate Parameters executes the command and uses its standard output as the default value for the parameter.

.tty
A name type parameter is similar to a string except that embedded white-space is not allowed. NOTE: this parameter is also a list, meaning that it can be specified multiple times and that each value is pushed onto a Perl LIST variable. In general you should quote all list elements. All types except switch may be 'list of'.

end-of-MM

```
Evap \@PDT, \@MM; # evaluate parameters
print "\nProgram name:\n $Options('help')\n\n";
if (defined $Options('verbose')) {print "Inverbose = $Options('verbose')\n\n";}
print "command = \"$Options('command')\"\n\n";
print "scale.factor = $Options('scale.factor')\n\n";
print "millisecond_update_interval = $Options('millisecond_update_interval')\n\n";
print "ignore_output_file_column_one = $Options('ignore_output_file_column_one')\n\n";
print "output = $Options('output')\n\n";
print "queue = $Options('queue')\n\n";
print "destination = $Options('destination')\n\n";
print "list of tty = \"\", join(' ', @($Options('tty'))), \"\"\n\n";
print "\nFile names:\n ", join ' ', @ARGV, "\n" if @ARGV;
```

Using the PDT as a guide, Evaluate Parameters parses a user's command line, returning the results of the evaluation to global variables of the form B<opt_parameter>, B<opt_parameter>, B<options('parameter')> or B<options('parameter')>, where I<parameter> is the full spelling of the command line parameter.

Of course, you can specify where you want Evaluate Parameters to return its results, in which case this historical feature of writing into your namespace is disabled.

Every command using Evaluate Parameters automatically has a I<-help> switch which displays parameter help; no special code is required in your application.

B<*> Customization of Evap's Help Output **>

There are several Help Hook strings that can be altered to customize B<Evap>'s help output. Currently there is only one general area that can be customized: usage and error text dealing with the trailing file list. For instance, if a command requires one or more trailing file names after all the command line switches, the default I<-help> text is:

file(s) required by this command

Some commands do not want trailing "file names", but rather some other type of information. An example is I<display_command_information> where a single Program_Name is expected. The following code snippet shows how to do this:

```
$setopt::Evap::evap_help_hooks('P_HHURFL') = " Program_Name\n";
$setopt::Evap::evap_help_hooks('P_HHERFL') =
    "\nA Program_Name is required by this command.\n\n";
$setopt::Evap::evap_help_hooks('P_HHERFL') =
    "\nA trailing Program_Name is required by this command.\n";
Evap \@PDT, \@MM;
```

As you can see, the hash B<evap_help_hooks> is indexed by a simple ordinal. The ordinals are shown below and are mostly self-explanatory. In case you don't have access to the source for Evaluate Parameters, here are the default values of the Help Hook strings.

```
$setopt::Evap::evap_help_hooks('P_HHURFL') = " file(s)\n";
$setopt::Evap::evap_help_hooks('P_HHUOFL') = " [file(s)]\n";
$setopt::Evap::evap_help_hooks('P_HHUNFL') = "\n";
$setopt::Evap::evap_help_hooks('P_HHERFL') =
    "\nfile(s) required by this command\n\n";
$setopt::Evap::evap_help_hooks('P_HHOFL') =
    "\n[file(s)] optionally required by this command\n\n";
$setopt::Evap::evap_help_hooks('P_HHENFL') = "\n";
$setopt::Evap::evap_help_hooks('P_HHERFL') =
    "Trailing file name(s) required.\n";
$setopt::Evap::evap_help_hooks('P_HHENFL') =
    "Trailing file name(s) not permitted.\n";
```

The Help Hooks naming convention is rather simple:

```
P_HHtf
P_HHtf
where:
P_HH implies an Evaluate Parameters Help Hook
t
type:
    U=Usage Help
    B=Brief and Full Help
    E=error message
f
file_list:
    RPL=required file_list
    OFL=optional file_list
    NFL=no_file_list
```


Note to I<genPerltk> and I<genTclTk> users: using these Help Hooks may cause the "genTk programs" to generate an unuseable Tk script. This happens because the "genTk programs" look for the strings "required by this command" or "optionally required by this command" in order to generate the file_list Entry widget - if these strings are missing the widget is not created. An easy solution is to ensure that your Help Hook text contains said string, just like the code snippet above; otherwise you must manually add the required Tk code yourself.

B<** Human Interface Guidelines **>

To make Evaluate Parameters successful, you, the application developer, must follow certain conventions when choosing parameter names and aliases.

Parameter names consist of one or more words, separated by underscores, and describe the parameter (for example, I<verbose> and I<spool_directory>).

You can abbreviate parameters: use the first letter of each word in the parameter name. Do not use underscores. For example, you can abbreviate I<command> as I<c> and I<delay_period> as I<dp>.

There are exceptions to this standard:

=over 4

=item *

I<password> is abbreviated I<pw>.

=item *

The words I<minimum> and I<maximum> are abbreviated I<min> and I<max>. So, the abbreviation for the parameter I<maximum_byte_count> is I<maxbc>.

=item *

There are no abbreviations for the parameters I<usage_help> and I<full_help>; I do not want to prevent I<uh> and I<fn> from being used as valid command line parameters.

=back

B<** Variables MANPAGER, PAGER and D_EVAP_DO_PAGE **>

The environment variable MANPAGER (or PAGER) is used to control the display of help information generated by Evaluate Parameters. If defined and non-null, the value of the environment variable is taken as the name of the program to pipe the help output through. If no paging program is defined then the program I<more> is used.

The boolean environment variable D_EVAP_DO_PAGE can be set to FALSE/NO/OFF/0, any case, to disable this automatic paging feature (or you can set your paging program to I<cat>).

B<** Return Values **>

B<EvaP()> behaves differently depending upon whether it's called to parse an

EvaP.pm

application's command line, or as an embedded command line parser (for instance, when using B<evap_pac()>).

Application Command Line	Embedded Command Line
error	exit(1)
success	return(0)
help	return(1)
	exit(0)
	return(-1)

=head1 SEE ALSO

```

evap(2)
evap.c(2)
EvaP.pm(2)
evap.tcl(2)
evap_pac(2)
addmm, add_message_modules(1)
disci, display_command_information(1)
genmp, generate_man_page(1)
genpdt, generate_pdt(1)
genPerltk, generate_Perltk_program(1)
genTclTk, generate_TclTk_program(1)

```

All available from directory ftp.Lehigh.EDU:/pub/evap/evap-2.x.

=head1 AUTHOR

Stephen O. Lidie <lusol@Lehigh.EDU>

=head1 HISTORY

```

lusol@Lehigh.EDU 94/10/28 (PDT version 2.0) Version 2.2
. Original release - derived from evap.pl version 2.1.
. Undef option values for subsequent embedded calls.

```

```

lusol@Lehigh.EDU 95/10/27 (PDT version 2.0) Version 2.3.0
. Be a strict as possible.
. Revert to -h alias rather than -?. (-? -?? -??? still available.)
. Move into Getopt class.
. Format for 80 columns (mostly).
. Optional third argument on EvaP call can be a reference to your own %options hash. If specified, the variables %options, %options and $opt* are not used.

```

```

lusol@Lehigh.EDU 97/01/12 (PDT version 2.0) Version 2.3.1
. Fix Makefile.PL so it behaves properly. Convert nroff man data to pod format.

```

=head1 COPYRIGHT

Copyright (C) 1993 - 1997 Stephen O. Lidie. All rights reserved.

This program is free software; you can redistribute it and/or modify it under the same terms as Perl itself.

=cut

require 5.002;
use English;

EvaP.pm

Evap.pm

Evap.pm

```

$P_ALIAS($parameter) = $alias; # remember alias
evap_PDT_error "Cannot have 'list of switch': \"$option\".\n"
if $P_INFO($parameter) =~ /^w1$/;

if ($default_value ne "" and $default_value ne '$required') {
    $default_value = $ENV{$P_ENV($parameter)} if $P_ENV($parameter)
    and $ENV{$P_ENV($parameter)};
    $P_DEFAULT_VALUE($parameter) = $default_value;
    evap_set_value 0, $type, $list, $default_value, $parameter;
} elsif ($evap_embed) {
    no strict 'refs';
    undef ${"${pkg}::opt_${parameter}"} if not defined $ref_Opt;
}

} # forend OPTIONS

if ($error) {
    print STDERR "Read the 'man' page \"EvaP.pm\" for details on PDT "
    "syntax.\n";
    exit 1;
}

# Process arguments from the command line, stopping at the first parameter
# without a leading dash, or a --. Convert a parameter alias into its full
# form, type-check parameter values and store the value into global
# variables for use by the caller. When complete call evap_fin to
# perform final processing.

```

ARGUMENTS:

```

while ($#ARGV >= 0) {
    $option = shift @ARGV; # get next command line parameter
    $value = undef; # assume no value

    $full_help = 1 if $option =~ /^-(full_help|Q??E)$/;
    $usage_help = 1 if $option =~ /^-(usage_help|Q??E)$/;
    $option = '-help' if $full_help or $usage_help or
    $option =~ /^-(Q??E)$/;

    if ($option =~ /^(-|--)/) { # check for end of parameters
        if ($option eq '--') {
            return evap_fin;
        }
        $option = $POSTMATCH; # option name without dash
    } else {
        unshift @ARGV, $option;
        return evap_fin;
    }

    foreach $alias (keys %P_ALIAS) { # replace alias with the full spelling
        $option = $alias if $option eq $P_ALIAS{$alias};
    }

    if (not defined($rt = $P_INFO($option))) {
        $found = 0;
        foreach $key (keys %P_INFO) { # try substring match
            if ($option eq substr $key, 0, $length) {
                if ($found) {

```

EvaP.pm

```

print STDERR "Ambiguous parameter: -$option.\n";
$error++;
last;
}
$found = $key; # remember full spelling
}
} # forend
$option = $found ? $found : $option;
if (not defined($rt = $P_INFO($option))) {
    print STDERR "Invalid parameter: -$option.\n";
    $error++;
    next ARGUMENTS;
}
} # ifend non-substring match

($required, $type, $list) = ($rt =~ /$pdt_reg_exp1/);

if ($type =~ /\w$/) {
    if ($#ARGV < 0) { # if argument list is exhausted
        print STDERR "Value required for parameter -$option.\n";
        $error++;
        next ARGUMENTS;
    } else {
        $value = shift @ARGV;
    }
}

if ($type =~ /\w$/) { # switch
    $value = 1;
} elsif ($type =~ /\d$/) { # integer
    if ($value =~ /^[+-]?[0-9]+$/) {
        print STDERR "Expecting integer reference, found \"$value\" "
        "for parameter -$option.\n";
        $error++;
        undef $value;
    }
} elsif ($type =~ /\r$/) { # real number, int is also ok
    if ($value =~ /^[+-]?(\d+(\.\d*)?)|\.\d+(\.\d*)?$/) {
        print STDERR "Expecting real reference, found \"$value\" "
        "for parameter -$option.\n";
        $error++;
        undef $value;
    }
} elsif ($type =~ /\$|\^|\$|^\$/) { # string or name or application
} elsif ($type =~ /\f$/) { # file
    if (length $value > 255) {
        print STDERR "Expecting file reference, found \"$value\" "
        "for parameter -$option.\n";
        $error++;
        undef $value;
    }
} elsif ($type =~ /\b$/) { # boolean
    $value = tr/a-zA-Z/;
    if ($value =~ /$pdt_reg_exp2/$pdt_reg_exp3/i) {
        print STDERR "Expecting boolean reference, found "
        "\"$value\" for parameter -$option.\n";
        $error++;
        undef $value;
    }
} elsif ($type =~ /\k$/) { # keyword

```

EvaP.pm

First try exact match, then substring match.

```
undef $found;
@keys = split ' ', $P_VALID_VALUES($option);
for ($i = 0; $i <= $#keys and not defined $found; $i++) {
    $found = 1 if $value eq $keys[$i];
}
if (not defined $found) { # try substring match
    $length = length $value;
    for ($i = 0; $i <= $#keys; $i++) {
        if ($value eq substr $keys[$i], 0, $length) {
            if (defined $found) {
                print STDERR "Ambiguous keyword for parameter "
                    . "-$option: $value.\n";
                $error++;
                last; # for
            }
            $found = $keys[$i]; # remember full spelling
        }
    } # forend
    $value = defined($found) ? $found : $value;
} # ifend
if (not defined $found) {
    print STDERR "\"$value\" is not a valid value for the "
        . "parameter -$option.\n";
    $error++;
    undef $value;
} # ifend type-check

next ARGUMENTS if not defined $value;

$list = '2' if $list =~ /^1$/; # advance list state
evap_set_value 1, $type, $list, $value, $option if defined $value;
# Remove from $required list if specified.
@P_REQUIRED = grep $option ne $_, @P_REQUIRED;
$P_INFO($option) = $required . $type . '3' if $list;
} # whilend ARGUMENTS

return evap_fin;
} # end evap

sub evap_fin {
    # Finish up Evaluate Parameters processing:
    # # If -usage_help, -help or -full_help was requested then do it and exit.
    # # Else,
    # # . Store program name in 'help' variables.
    # # . Perform deferred evaluations.
    # # . Ensure all $required parameters have been given a value.
    # # . Ensure the validity of the trailing file list.
    # # . Exit with a Unix return code of 1 if there were errors and
    # # $evap_embed = 0, else return to the calling Perl program with a
    # proper return code.
```

use File::Basename;

```
my($m, $p, $required, $type, $list, $def, $rt, $def, $element, $is_string,
    $pager, $do_page);

# Define Help Hooks text as required.

$evap_help_hooks{'P_HHURFL'} = " file(s)\n"
    if not defined $evap_help_hooks{'P_HHURFL'};
$evap_help_hooks{'P_HHUOFL'} = " [file(s)]\n"
    if not defined $evap_help_hooks{'P_HHUOFL'};
$evap_help_hooks{'P_HHUNFL'} = "\n"
    if not defined $evap_help_hooks{'P_HHUNFL'};
$evap_help_hooks{'P_HHBRFL'} = "\nfile(s) required by this command\n\n"
    if not defined $evap_help_hooks{'P_HHBRFL'};
$evap_help_hooks{'P_HHBOFL'} = "\n[file(s)] optionally required by this comm
and\n\n"
    if not defined $evap_help_hooks{'P_HHBOFL'};
$evap_help_hooks{'P_HHENFL'} = "\n"
    if not defined $evap_help_hooks{'P_HHENFL'};
$evap_help_hooks{'P_HHERFL'} = "Trailing file name(s) required.\n"
    if not defined $evap_help_hooks{'P_HHERFL'};
$evap_help_hooks{'P_HHENFL'} = "Trailing file name(s) not permitted.\n"
    if not defined $evap_help_hooks{'P_HHENFL'};

my $want_help = 0;
if (defined $ref_opt) {
    $want_help = $ref_opt->{'help'};
} else {
    no strict 'refs';
    $want_help = ${pkg}::opt_help;
    $want_help = $$want_help;
}

if ($want_help) { # see if help was requested
    my($optional);
    my($parameter_help_in_progress) = 0;
    my($type_list) = (
        'w' => 'switch',
        'i' => 'integer',
        's' => 'string',
        'r' => 'real',
        'f' => 'file',
        'b' => 'boolean',
        'k' => 'key',
        'n' => 'name',
        'a' => 'application',
    );
    # Establish the pager and open the pipeline. Do no paging if the
    # boolean environment variable D_EVAP_DO_PAGE is FALSE.

    $pager = 'more';
    $pager = $ENV{'PAGER'} if defined $ENV{'PAGER'} and $ENV{'PAGER'};
    $pager = $ENV{'MANPAGER'} if defined $ENV{'MANPAGER'} and
        $ENV{'MANPAGER'};
    $pager = '|' . $pager;
    if (defined $ENV{'D_EVAP_DO_PAGE'}) and
```

```
((do_page = ENV{'D_EVAP_DO_PAGE'}) ne '') {
    do_page = tr/a-zA-Z/;
    spager = '>-' if $do_page =~ /$pdt_reg_exp3/;
}
open PAGER, "$spager";

print PAGER "Command Source: $PROGRAM_NAME\n\n" if $full_help;

# Print the Message Module text and save any full help. The key is the
# parameter name and the value is a list of strings with the newline as
# a separator. If there is no Message Module or it's empty then
# display an abbreviated usage message.

if ($usage_help or not defined ${lref_MM} or ${lref_MM} < 0) {
    $basename = basename($PROGRAM_NAME, "");
    print PAGER "usage: ", $basename;
    $optional = '';
    foreach $p (@P_PARAMETERS) {
        if ($p_INFO{$p} =~ /^R.?$/) { # if $required
            print PAGER " -$P_ALIAS($p)";
        } else {
            $optional .= " -$P_ALIAS($p)";
        }
    }
    print PAGER " [$optional]" if $optional;
    if ($file_list =~ /$pdt_reg_exp5/) {
        print PAGER "$evap_Help_Hooks{'P_HHUFEL'}";
    } elsif ($file_list =~ /$pdt_reg_exp6/) {
        print PAGER "$evap_Help_Hooks{'P_HHUFEL'}";
    } else {
        print PAGER "$evap_Help_Hooks{'P_HHUNFL'}";
    }
} else {
    MESSAGE_LINE:
    foreach $m (@{$lref_MM}) {
        if ($m =~ /\.\.(.*)$/) { # look for 'dot' leadin character
            $p = $1; # full spelling of parameter
            $parameter_help_in_progress = 1;
            $parameter_help($p) = "\n";
            next MESSAGE_LINE;
        } # ifend start of help text for a new parameter
        if ($parameter_help_in_progress) {
            $parameter_help($p) .= $m . "\n";
        } else {
            print PAGER $m, "\n";
        }
    } # forend MESSAGE_LINE
} # ifend usage_help

# Pass through the PDT list printing a standard evap help summary.
print PAGER "Parameters:\n";
if (not $full_help) (print PAGER "\n");
```

EvaP.pm

```
ALL PARAMETERS:
foreach $p (@P_PARAMETERS) {
    no strict 'refs';
    if ($full_help) (print PAGER "\n");
    if ($p =~ /^help$/) {
        print PAGER "-$p, $P_ALIAS($p), usage_help, full_help: Display C
        command information\n";
        if ($full_help) {
            print PAGER <<"end of DISCI";
            \nDisplay information about this command, which includes
            \ta command description with examples, plus a synopsis of
            \the command line parameters. If you specify -full_help
            \rather than -help complete parameter help is displayed
            \tif it's available.
            end_of_DISCI
        }
        next ALL_PARAMETERS;
    }
    $str = $p_INFO{$p}; # get encoded required/type information
    ($required, $type, $list) = ($str =~ /$pdt_reg_exp1/); # unpack
    $type = $type_list($type);
    $sis_string = ($type =~ /^strings$/);
    print PAGER "-$p, $P_ALIAS($p): ", $list ? 'list of ' : '', $type;
    print PAGER " ", join(' ', split(' ', $P_VALID_VALUES($p))),
        ", keyword" if $type =~ /^key$/;
    my($sref);
    if (defined $lref_Opt) {
        $sref = $lref_Opt->($p);
        $sref = \@{$lref_Opt->($p)} if $list;
    } else {
        $sref = "${pkg}::opt_${p}";
    }
    if ($list) {
        if ($def = defined ${sref} ? 1 : 0;
        ) else {
            $def = defined ${sref} ? 1 : 0;
        }
    }
    if ($required =~ /\O$/ or $def == 1) { # if $optional or defined
        if ($def == 0) { # undefined and $optional
            print PAGER "\n";
        } else {
            # defined (either $optional or $required), displ
            ay the default value(s)
            if ($list) {
                print PAGER $P_ENV{$p} ? " = $P_ENV{$p}, " : " = ";
                print PAGER $sis_string ? "(" : "(", $sis_string ? join(
                ",", "", @{$sref}), @{$sref}), "\n";
            } else {
                # not 'list of'
                print PAGER $P_ENV{$p} ? " = $P_ENV{$p}, " : " = ";
                print PAGER $sis_string ? "\n" : "\n";
            }
        }
    }
    ? "\n" : "\n";
    } # ifend 'list of'
```

EvaP.pm

```

    } # ifend
  ) elseif ($required =~ /R/) {
    print PAGER "$P_ENV($p) ? " = $P_ENV($p), " : " = ";
    print PAGER "\$required\n";
  } else {
    print PAGER "\n";
  } # ifend Optional or defined parameter

  if ($full_help) {
    if (defined $parameter_help($p)) {
      print PAGER "$parameter_help($p)";
    } else {
      print PAGER "\n";
    }
  }
} # forend ALL_PARAMETERS

if ($file_list =~ /$pdt_reg_exp5/) {
  print PAGER "$evap_help_hooks('P_HHBOFL')";
} elseif ($file_list =~ /$pdt_reg_exp6/) {
  print PAGER "$evap_help_hooks('P_HHBRFL')";
} else {
  print PAGER "$evap_help_hooks('P_HHENFL')";
}

close PAGER;
if ($evap_embed) {
  return -1;
} else {
  exit 0;
}

} # ifend help requested

# Evaluate remaining unspecified command line parameters. This has been
# deferred until now so that if -help was requested the user sees
# unevaluated boolean, file and backticked values.

foreach $parameter (@P_PARAMETERS) {
  if (not $P_EVALUATE($parameter) and $P_DEFAULT_VALUE($parameter)) {
    ($required, $type, $list) = ($P_INFO($parameter) =~ /$pdt_reg_exp1/)
    ;
    if ($type ne 'w') {
      $list = 2 if $list; # force re-initialization of the list
      evap_set_value 1, $type, $list, $P_DEFAULT_VALUE($parameter), $p
      parameter;
    } # ifend non-switch
  } # ifend not specified
} # forend all PDT parameters

# Store program name for caller.
evap_set_value 0, 'w', '', $PROGRAM_NAME, 'help';

# Ensure all $required parameters have been specified on the command line.
foreach $p (@P_REQUIRED) {
  print STDERR "Parameter $p is required but was omitted.\n";
}

```

```

    $error++;
  } # forend

# Ensure any required files follow, or none do if that is the case.
if ($file_list =~ /$pdt_reg_exp4/ and $#ARGV > 0 - 1) {
  print STDERR "$evap_help_hooks('P_HHENFL')";
  $error++;
} elseif ($file_list =~ /$pdt_reg_exp6/ and $#ARGV == 0 - 1) {
  print STDERR "$evap_help_hooks('P_HHBRFL')";
  $error++;
}

print STDERR "Type $PROGRAM_NAME -h for command line parameter "
  "information.\n" if $error;

exit 1 if $error and not $evap_embed;
if (not $error) {
  return 1;
} else {
  return 0;
}

} # end evap_fin

sub evap_PDT_error {
  # Inform the application developer that they've screwed up!

  my($msg) = @ARG;

  print STDERR "$msg";
  $error++;
  next OPTIONS;
} # end evap_PDT_error

sub evap_set_value {
  # Store a parameter's value; some parameter types require special type
  # conversion. Store values the old way in scalar/list variables of the
  # form $opt_parameter and @opt_parameter, as well as the new way in hashes
  # named %options and %options. 'list of' parameters are returned as a
  # reference in %options/%options (a simple list in @opt_parameter). Or,
  # just stuff them in a user hash, is specified.
  # Evaluate items in grave accents (backticks), boolean and files if
  # 'evaluate' is TRUE.
  # Handle list syntax (item1, item2, ...) for 'list of' types.
  # Lists are a little weird as they may already have default values from the
  # PDT declaration. The first time a list parameter is specified on the
  # command line we must first empty the list of its default values. The
  # P_INFO list flag thus can be in one of three states: 1 = the list has
  # possible default values from the PDT, 2 = first time for this command
  # line parameter so empty the list and THEN push the parameter's value, and
  # 3 = just keep pushing new command line values on the list.

  my($evaluate, $type, $list, $v, $hash_index) = @ARG;

```


Mail for Adam Sartiel

Sun Aug 9 10:52:29 1998

```

) elseif (defined $alias($PROGRAM_NAME)) {
    $proc = $alias($PROGRAM_NAME);
} else {
    print STDERR <<"end of ERROR";
    Error - unknown command '$PROGRAM_NAME'. Type 'disac -do f' for a
    list of valid application commands. You can then ... \n
    Type 'xyzy -h' for help on application command 'xyzy'.
    end_of_ERROR
    next GET_USER_INPUT;
}

if ($PROGRAM_NAME eq '!') {
    @ARGV = $args;
} else {
    @ARGV = &shellwords($args);
}

eval "$$proc"; # call the evap/user procedure
print STDERR $EVAL_ERROR if $EVAL_ERROR;

} # whileend GET_USER_INPUT
continue { # while GET_USER_INPUT
    print STDOUT "$prompt";
} # continuend
print STDOUT "\n" unless $prompt eq "";
} # end evap_pac

sub evap_bang_proc {
    # Issue commands to the user's shell. If the SHELL environment variable is
    # not defined or is empty, then /bin/sh is used.

    my $cmd = $ARGV[0];

    if ($cmd ne '') {
        evap_setup_for_evap 'bang' unless defined @bang_proc_PDT;
        $evap_help_hooks{'P_HHUOFL'} = "Command(s)\n";
        $evap_help_hooks{'P_HHBOFL'} = "\nA list of shell Commands.\n\n";
        my $junk = \@bang_proc_MM; # suppress -w warning
        if (Evap(\@bang_proc_PDT, \@bang_proc_MM) != 1) {return;}
        system "$shell -c '$cmd'";
    } else {
        print STDOUT "Starting a new '$shell' shell; use 'exit' to return .
        "to this application.\n";
        system $shell;
    }
}

} # end evap_bang_proc

sub evap_disac_proc {
    # Display the list of legal application commands.

    my(%commands) = @ARGV;
    my(@brief, @full, $name, $long, $alias);

    evap_setup_for_evap 'disac' unless defined @disac_proc_PDT;
    my $junk = \@disac_proc_MM; # suppress -w warning
    if (Evap(\@disac_proc_PDT, \@disac_proc_MM) != 1) {return;}

```

EvaP.pm

Mail for Adam Sartiel

Sun Aug 9 10:52:29 1998

```

foreach $name (keys %commands) {
    if ($name =~ /\|/) {
        ($long, $alias) = ($name =~ /(.*?)\|(.*?)/);
    } else {
        $long = $name;
        $alias = '';
    }
    push @brief, $long;
    push @full, ($alias ne '') ? "$long, $alias" : "$long";
}

open H, ">$Options{'output'}";
if ($Options{'display_option'} eq 'full') {
    print H "\nFor help on any application command (or alias) use the -h swi
    tch. For example,\n";
    print H "try 'disac -h' for help on 'display.application.commands'.\n";
    print H "\nCommand and alias list for this application:\n\n";
    print H " ", join("\n ", sort(@full)), "\n";
} else {
    print H join("\n", sort(@brief)), "\n";
}
close H;

} # end evap_disac_proc

sub evap_setup_for_evap {
    # Initialize evap_pac's builtin commands' PDT/MM variables.

    my($command) = @ARGV;

    open IN, "ar p $message_modules {(command)_pdt}";
    eval "\${command}_proc_PDT = <IN>";
    close IN;

    open IN, "ar p $message_modules {(command)_mm}";
    eval "\${command}_proc_MM = grep \$@ = s/\n\$/ /, <IN>";
    close IN;

} # end evap_setup_for_evap
1;

```

EvaP.pm

A-392

The following public domain programs are used:

1. BLASTN 2.0.3 [Nov-14-1997] - from NCBI, NIH.
2. perl, version 5.004_01, including the following includes from perl:
 - Shell
 - integer
 - FileHandle
 - strict
 - Carp

All the above are available through regular PERL distribution, and from the CPAN public domain.

Clustering instructions

Written by Raveh Gill-More, 26.2.98

Updated by Avi Rosenberg, 28.5.98

- The clustering should be done with the user *prod*.
 - The executables are in /home/prod/bin/ARCH. ARCH is the architecture of the machine being used - can be either DEC, SUN or SGI (but the last one isn't recommended, and many programs don't exist there).
 - It is recommended to do all of the script work on bioserv (since the data itself is on bioserv, and also, bioserv has a lot of memory - 1GB). The only program that MUST run on a dec is the cluster program.
1. FTP: Download GenBank. These are all the files located at ncbi.nlm.nih.gov/genbank. whose names are of the form gb*.seq.Z. A new version should appear around the 15th of every even month. In version 105 there are 20 EST files and 17 other files. The download process is usually carried out in two stages:
 - a. Download to Boston (cgen.com). You need a login. There should be room in /gate3/tmp (especially after you remove the old GenBank version from there). Takes ~3 hours.
 - b. Make a directory called prodgb??? (according to the current version name) on a disk with a lot of free space (over 20GB recommended). This should be done on a raid disk, connected to bioserv (in the rest of the document this will be called \$BASEDIR. In it make directories named: mouse, human, and in each one, make directories named cluster and assembly. This can be done prior to the release of GenBank. It is recommended to do:

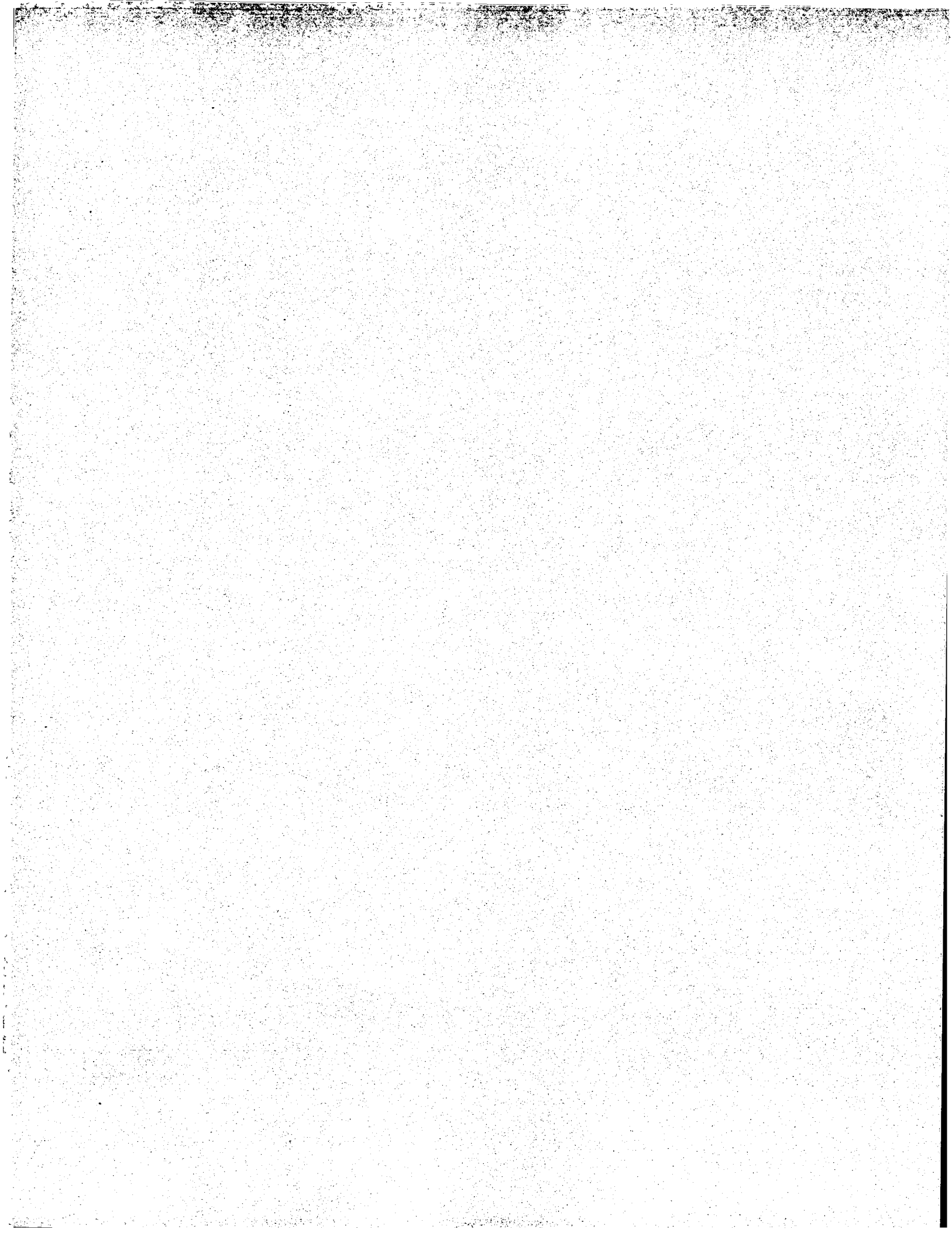

```
setenv BASEDIR /dir/bioserv.....
```
 - c. In the directory /dir/bioserv2/databases/..... make a directory In the rest of the document this will be called \$GENBANKDIR. It is recommended to do:


```
setenv GENBANKDIR /dir/bioserv.....
```
 - c. Download to Israel (to the /dir/bioserv2/databases....) from Boston using the ISDN lines. Consult Artur about that. Should take ~10 hours.
 2. UNCOMPRESS: Uncompress the necessary files. You need the gbest* files, gbprod.seq.Z, gbpril.seq.Z and gbpri2.seq.Z.


```
cd $GENBANKDIR
foreach f (gbest* gbpri* gbprod*)
    echo $f
    gunzip $f
end
```
 3. FETCH: We need to fetch all the sequences of the organism we are interested in. This can be done by running:

```
fetch.pl organism=[M|H] molecule=[E|R] in= out=
```

The first letter denotes the organism (Human or Mouse), the second is the sequence-type (Est or Rna), next comes the name of the GenBank file, followed by the output file name for the FastA format. You should work in two directories : "mouse" and "human". The commands are:



```

for mouse:
cd $GENBANKDIR
fetch.pl organism=M molecule=R in= gbrod.seq out= $BASEDIR/mouse/cluster/rna
foreach i (`ls -l gbest*.seq | sed 's/gbest//g' | sed 's/.seq//g' `)
    fetch.pl organism=M molecule=E in= gbest${i}.seq out= $BASEDIR/mouse/clus
end

for human:
cd $GENBANKDIR
foreach i (`ls -l gbpri*.seq | sed 's/gbpri//g' | sed 's/.seq//g' `)
    fetch.pl organism=H molecule=R in= gbpri${i}.seq out= $BASEDIR/human/clus
end
foreach i (`ls -l gbest*.seq | sed 's/gbest//g' | sed 's/.seq//g' `)
    fetch.pl organism=H molecule=E in= gbest${i}.seq out= $BASEDIR/human/clus
end

```

If you want to speed up the process, you can split up the fetching of the est files to work on several computers.

4. MERGE: The last stage gives you 2 rna files (one for mouse and one for human), and many est files for both mouse and human. We need to combine the est files into a smaller number of files (to simplify the rest of the process), but each file should not exceed the size of about 250MB. we will merge these files using the script merge_and_divide_files.pl:

```

cd $BASEDIR/mouse/cluster
merge_and_divide_files.pl outprefix=est max_size=250000000 file_type=fasta t

cd $BASEDIR/human/cluster
merge_and_divide_files.pl outprefix=est max_size=250000000 file_type=fasta t

```

After checking that all the files are ok (the sum of the sizes of the tmp.est* files and the est* files are the same), all temporary files can be deleted.

5. DO_BLAST: Our sequences have to be scanned for repeats, vectors and highly expressed genes. This is done using BLAST2. A script running BLAST and parsing the results is called "do_blast_cln.pl". Prepare the directory "blast_res" under the cluster directory of the human and mouse, and in it create the following directories:

```

for mouse:
cd $BASEDIR/mouse/cluster
foreach f (est? rna)
    foreach type (r v a)
        mkdir -p blast_res/${f}_${type}
    end
end

for human:
cd $BASEDIR/human/cluster
foreach f (est? rna)
    foreach type (r v a)
        mkdir -p blast_res/${f}_${type}
    end
end

```

In each of these directories, one scan will be run. The directory will end up containing one blast2 output file per query sequence (vector, repeat or abundantly occurring gene). Additionally, the files in each directory will

A-396

be parsed to produce the necessary annotation.

Now you need to format your data. Run:

```
cd $BASEDIR/mouse/cluster
formatdb -i rna -p F
foreach f (est*)
  formatdb -i $f -p F
end
cd $BASEDIR/human/cluster
formatdb -i rna -p F
foreach f (est*)
  formatdb -i $f -p F
end
```

For each FastA file you should get 3 files containing formatted data, with suffixes ".nin", ".nsq" and ".nhr".

The commands for mouse are:

```
cd $BASEDIR/mouse/cluster
foreach f (est? rna)
  cd blast_res/${f}_r
  do_blast_cln.pl ../../$f /dir/bioserv2/databases/repeat/mouse/mouse_repea
  cd ../../${f}_v
  do_blast_cln.pl ../../$f /dir/bioserv2/databases/vector/vector ../../${f}
  cd ../../${f}_a
  do_blast_cln.pl ../../$f /dir/bioserv2/databases/abundant/mouse/mouse_abu
  cd ../../
end
```

The commands for human are:

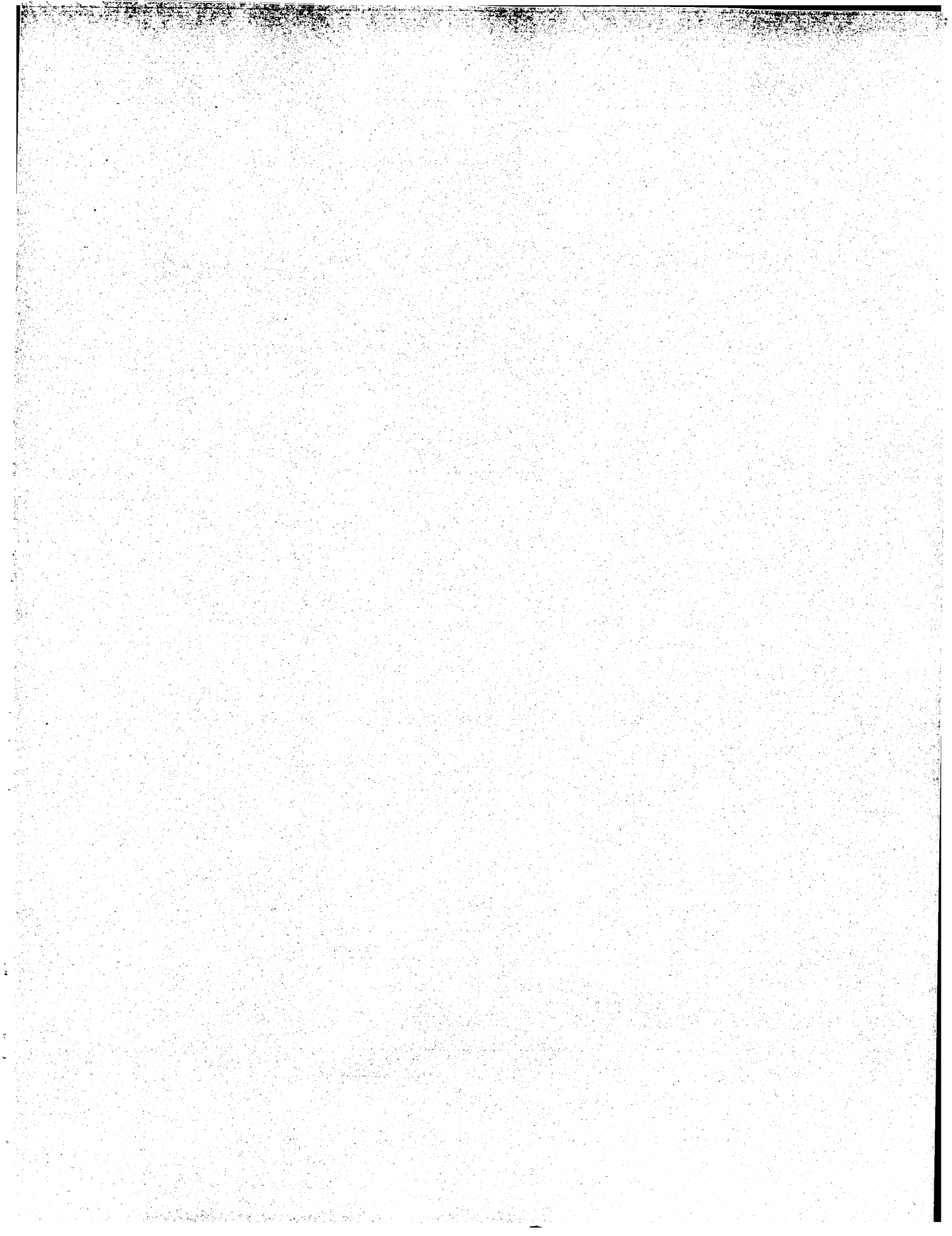
```
cd $BASEDIR/human/cluster
foreach f (est? rna)
  cd blast_res/${f}_r
  do_blast_cln.pl ../../$f /dir/bioserv2/databases/repeat/human/human_repea
  cd ../../${f}_v
  do_blast_cln.pl ../../$f /dir/bioserv2/databases/vector/vector ../../${f}
  cd ../../${f}_a
  do_blast_cln.pl ../../$f /dir/bioserv2/databases/abundant/human/human_abu
  cd ../../
end
```

This takes a lot of time (especially the human ones...), so you might want to do this on different computers (i.e. run each do_blast_cln.pl command on a different computer, without the foreach loop).

You can then remove the formatted databases (rm *.nhr *.nin *.nsq), and keep just the *.?bic files. In each file are 6 columns: the name of the vector/repeat/gene that was found, the query it hit, the strand, the score and the start and end position of the hit in the query.

REMARK: BLAST2 is very sensitive to the abundance of its query in the database. In some cases (such as ALU repeats) there is such a large number of hits that BLAST2 runs out of memory and crashes. This is the reason we keep the ests in a few separate files up to this stage. If we unite them before this stage - BLAST2 fails. Even so, BLAST2 takes a very long time and a lot of memory processing some of the more abundant repeats and genes. In the future it might be necessary to divide the human ESTs into more than two files.

To check if any of the BLAST2 runs failed look at the length of all ".tmp"



files which were created by the do_blast script. Sort them by length and see whether the shortest ones have terminated normally. All these files should be above 1K bytes long if they are OK. The command is:

```
ls -l *.tmp | sort -nr -k 5 | tail
```

If any of the files is incomplete - the BLAST would have to be re-run after dividing the appropriate EST file into two or more parts.

From now and on (until the assembly stage), you should work in the "cluster" directory of each organism.

6. MERGE: the pairs of EST bicfiles should now be merged while keeping the lines sorted by EST name:

```
sort -m -k 2 est?.abic > est.abic
sort -m -k 2 est?.rbic > est.rbic
sort -m -k 2 est?.vbic > est.vbic
```

The est?.[arv]bic files can now be safely deleted. This is also the time to merge the human EST files:

```
cat est? > est
```

and remove est? (after checking the file sizes to make sure all is well).

7. SORT_EST: The ESTs (mRNAs) should be sorted according to alphabetical order (to fit the order of lines in the bicfile). To do that run:

```
BigSort.pl in=rna out=rna.sorted -fasta keys="('^(\\S+)\\s')"
```

```
BigSort.pl in=est out=est.sorted -fasta keys="('^(\\S+)\\s')"
```

It is very advisable to run these commands on bioserv (i.e. locally on the machine that holds the files, and has a lot of memory).

Note: it is important to do the sorting to a new file and compare it's length to the original "est" file. Many things can go wrong while sorting, and you don't want to re-create the "est" file. Only after you verify the length of the two files are the same can you do:

```
mv est.sorted est
mv rna.sorted rna
```

8. CLN: the cleaning stage. Assuming you have named all the files as in the previous sections you should do:

```
cln dir=. in=rna -mrna
cln dir=. in=est
```

All other parameters shouldn't be touched, but here is a list of what they mean: The suffixes control the names of the cleaning output files. The "pbound" parameter is the bound for inserting annotation about repeats or vectors to the sequence header, and the rest of the numbers are parameters for the wraparound dynamic program which will try to find low-complexity regions in the sequence. It is very important to set the "-mrna" flag when cleaning mRNAs. Another parameter that exists is the "-no_cut_n" flag, which - if specified - doesn't cut the ends of the sequences based upon the number of N's in "windows" of the sequence.

Three output files are produced. They are (for the est file):

est.info - information on what was found and cleaned.
 est.xout - a crossed-out version of the clean data (for clustering)
 est.mask - an unmasked version (for assembly and later stages)
 The program also parses the header line and converts it to proper Rich-FastA format.

9. ABUNDANT: Now we throw away sequences which are connected to abundant genes. First we need a list of these sequences' names:

```
awk '$6 - $5 > 50 {print $2}' est.abic | uniq > est.ab.names
awk '$6 - $5 > 50 {print $2}' rna.abic | uniq > rna.ab.names
```

And now, we need to separate the abundant sequences from the rest:

```
get_ests.pl est.ab.names est.xout est.xout.ab est.xout.ok
get_ests.pl est.ab.names est.umsk est.umsk.ab est.umsk.ok
get_ests.pl rna.ab.names rna.xout rna.xout.ab rna.xout.ok
get_ests.pl rna.ab.names rna.umsk rna.umsk.ab rna.umsk.ok
```

10. CAT: the files "est.xout.ok" and "rna.xout.ok" can now be concatenated into one file (call it "seq"), which contains the data to be clustered:

```
cat rna.xout.ok est.xout.ok > seq
```

11. CLUSTER: Now it is time to cluster, or (actually) to contig. Use an alpha, with at least 256M of memory (currently mnm, godiva, tobler, bacio, panama and peru are good, but lindt and chile are better, since they have more memory). You might want to change the "dir" and "in" parameters (if they are not like the default value - dir=. and in=seq), and the "mem" parameter. Mem should be set to 50 for human, 80 for mouse. Also you can play with the depth, and choose at most one of the flags (usually -sw, to perform a Smith-Waterman check at the end of the bin clustering). So, the command you will end up typing is:

```
cluster mem=50 -sw >& stdout.cluster
```

The output is a pairs file (seq.pairs) containing for each matching pair of sequences, data about the match (start and end in each sequence and quality).

12. MAKE_NAMES: The next stage creates the contigs out of the pairs, and gives each contig the name of the oldest sequence in it. A program for this can be found in make_names. The only parameters are "dir" and "in":

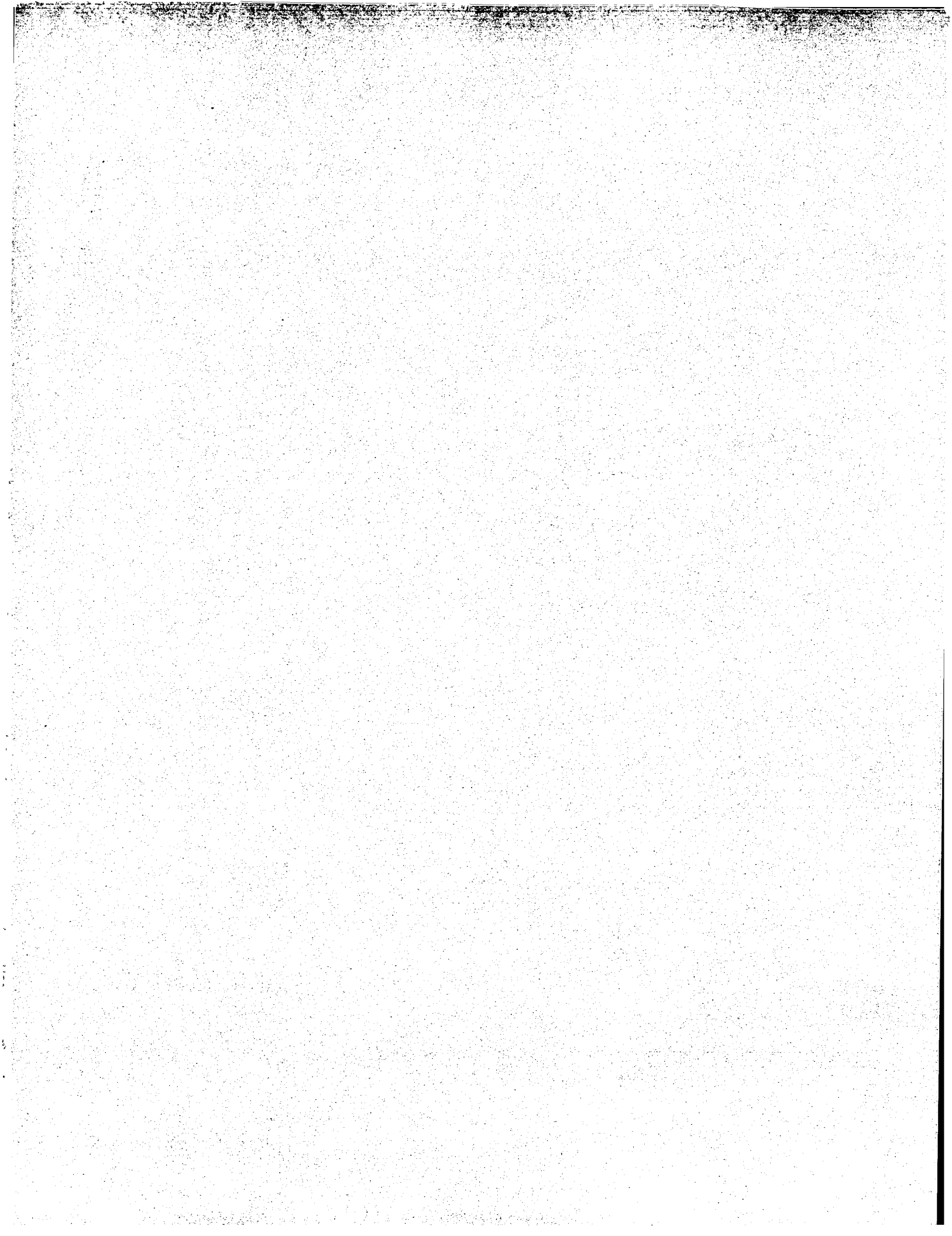
```
make_names dir=. in=seq
```

The output is a table with four columns: the first one is the sequence name (in order of appearance), then the contig name, which is identical to the name of the oldest sequence in the contig (and, if there are several sequences from the same old date - the first of them lexicographically), the date the sequence was entered into GenBank, and the size of the contig. This file will be updated later to contain more information.

13. STATISTICS: we can now find out the sizes of our contigs:

```
awk '{(print $2)} seq.clu | sort | uniq -c | awk '{(print $1)} | sort +nr > s
```

Now every line in "stat" holds the size of a single contig (sorted in descending order). See the statistics appendix at the end for instructions on how to obtain the relevant "clustering" numbers from this file.



A-399

14. GET_BIG_CONTIGS: The next few items are optional. If the biggest few contigs we found are VERY big, we would like to break them down further, by re-clustering all the sequences which participate in them, using more stringent conditions. In the case of mouse (ver 105) this was unnecessary, but in the case of human it was. This depends mainly on the comprehensiveness of the abundant gene database. So: if you decide not to recontig, you can go to item 19 (RICH_FASTA) otherwise, we start by keeping a version of the first clustering output:

```
cp seq.pairs seq.pairs.first
cp seq.clu seq.clu.first
cp stat stat.first
```

Now we have to get the names of the sequences we want to extract.
For human this is:

```
awk '$4 > 500' seq.clu | awk '{print $1}' > seq.big.names
```

For mouse use 200 instead of 500. We now extract the sequences themselves:

```
get_ests.pl seq.big.names seq seq.big
```

15. RECONTIG: Repeat stages 11, 12, 13 for "seq.big". In stage 11 add the flag "-only_full_overlap". You should get files "seq.big.pairs", "seq.big.clu" and "stat.big".
16. MERGE STATS: We need to merge several files which are results of our two clustering runs. We begin by merging the "stat" files:

```
awk '$1 <= 500' stat > tmp      (200 for mouse)
sort -m -nr tmp stat.big >! stat
```

Check everything is OK, and delete "stat.big" and "tmp".

17. MERGE CLU: We now merge the "seq.clu" files:

```
awk '$4 <= 500' seq.clu > tmp      (200 for mouse)
cat seq.big.clu tmp | sort -kb2 >! seq.clu
```

The resulting file "seq.clu" will be sorted by contigs in alphabetical order. Delete "seq.big.clu" and "tmp", after checking all is well.

18. MERGE PAIRS: Next, we merge the "seq.pairs" files. Note: this will overwrite the output of the clustering, which took several hours to produce. If there is any doubt we might need the original output - it should be copied to a different name before it is overwritten!

```
delete_big_pairs.pl seq.big.names seq.pairs seq.small.pairs
cat seq.big.pairs seq.small.pairs >! seq.pairs
```

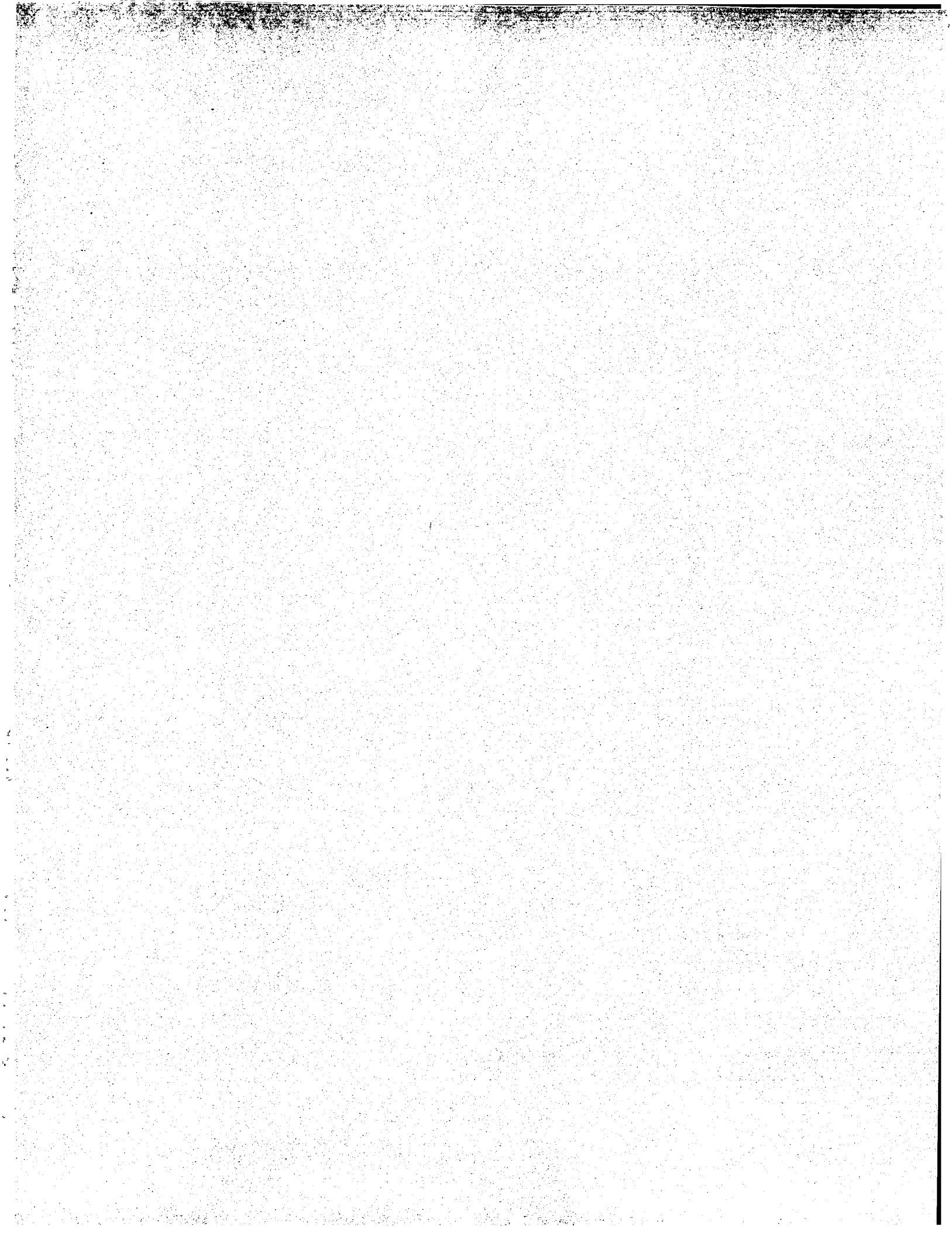
After checking, delete "seq.big.pairs" and "seq.small.pairs". At this stage you can also delete "seq" and "seq.big".

19. RICH FASTA: it is now (finally) time to create the rich fasta file that includes contig information though it will not be in its final form yet. Run:

```
make_fasta.pl seq.clu rna.umsk.ok est.umsk.ok CCH106_ rich_fasta
```

The parameter before last will act as a prefix to all contig names.

20. CLONE INFORMATION: We now add information from the clone annotation of



A-400

our input sequences, to unite contigs into clusters, where we have no direct evidence of sequence overlap, but where at least twice, lab technicians have assured us that a pair of sequences, one in each cluster, were produced while sequencing the same cDNA. We run:

```
clone_clusters1.pl rich_fasta 1000 2 tmp
clone_clusters2.pl tmp clone_info
```

the 1000 means we will not attempt to unite clusters bigger than 1000 sequences with other clusters (they tend to attract a lot of other clusters some of them erroneously). The 2 means we insist on 2 inter-cluster links before we believe the information (unless one of the cluster is a singleton, in which case one link is all we can get and it is considered to be enough). tmp is a temporary file name, which can be deleted when the second script has finished. The reason the stage is divided into 2 scripts is technical (memory allocation problems of perl).

21. CLONE_STATS: to obtain statistics about what the clone information can tell us we run:

```
awk '{print $3}' clone_info | uniq -c | awk '{print $1}' | sort +0nr > clon
```

Now in each line we have a cluster size (number of contigs contigs).

22. ADD CLONE INFORMATION: now we add cluster information, from file clone_info to seq.clu, and re-sort it by cluster name in alphabetical order:

```
add_clone_info.pl seq.clu clone_info seq.clu.new
sort -b -k 3,3 -k 2,2 -k 1,1 seq.clu.new >! seq.clu
```

After a check, you can delete "seq.clu.new".

The file now has 7 fields per line. They are: sequence name, contig name, cluster name, sequence date, contig size in sequences, cluster size in contigs and cluster size in sequences.

See the statistics appendix at the end for instructions on how to obtain the relevant "clone information" numbers.

23. SORT PAIRS: We now add the contig and cluster name to every line in "seq.pairs", and sort it by cluster name in alphabetical order:

```
sort_pairs.pl seq.clu seq.pairs seq.pairs.sorted
```

Then do 'mv seq.pairs.sorted seq.pairs'.

24. COMPLETE RICH FASTA: Now we need to add the cluster name to the rich-fasta file. This is done like this:

```
complete_rich_fasta.pl seq.clu rich_fasta rich_fasta.complete
```

On the mouse data this works fine. However, on the human data there is not enough memory to run the script in one go. The trick here is to divide the file "seq.clu" into two or more parts. To do that two additional parameters (first and last) may be supplied to the script, telling it which lines of "seq.clu" to use. So the whole thing should be:

```
complete_rich_fasta.pl seq.clu rich_fasta rich_fastal 0 500000
complete_rich_fasta.pl seq.clu rich_fastal rich_fasta.complete 500001
```

and then, after checking everything, do:

A-401

```
rm rich_fastal
```

25. SORT: the Rich-FastA file needs to be sorted. We will use BigSort.pl again, but this time with a different definition of the keys:

```
BigSort.pl in=rich_fasta.complete out=rich_fasta -fasta keys="('\#CU\s+CC...
```

It is very advisable to run this command on bioserv (i.e. locally on the machine that holds the files, and even more important, has a lot of memory).

Check that rich_fasta.complete and rich_fasta are of the same size exactly, and if they are you can do:

```
rm rich_fasta.complete
```

26. ASSEMBLY INPUT: we have two files which are passed to the assembly. They are called "rich_fasta" (Rich-FastA format, sorted by clusters and then by clones), and "seq.pairs" (also sorted, each line contains match information between two sequences). The first step is to get all of the sequences in the cluster to be in the same direction (hopefully the right one). This is done using the flipper program:

```
flipper gb_ver = your_version dir=. in= rich_fasta pairs_file= seq.pairs ou
```

27. ASSEMBLY: Since the assembly takes quite a lot of time, it is advisable to split up the work between some machines. To do this, we prepare a list of contigs that each machine will work on, taking into consideration the proportional strength of the machine (the DECs are about 2-2.5 times faster than SUNs):

```
SplitClusters.pl in=seq.clu config_file= assembly_config_file
```

The file assembly_config_file should be in the following format

```
# any line beginning with a # is a comment
#
# lines of the format [A-Z]+= \d+ are the relative strength of the
# types of the machine
# lines of the format [a-z]+\s+[A-Z]+ are the names of the machines and
# their types
# for example:
DEC=2
SUN=1
# Note that the machine types MUST be in upper-case
```

```
bacio      DEC
godiva     DEC
mnm        DEC
lindt      DEC
peru       DEC
panama     DEC
bioserv1   SUN
bioserv2   SUN
mir        SUN
```

```
# means that DECs are twice as strong as SUNs, and we will use
# 6 DEC machines (named bacio, godiva, mnm, lindt, peru, panama),
# and 3 SUN machines (named bioserv1, bioserv2, mir).
```

The output of this are lists named .list.

A-402

Make a directory on each machine that you wish to run, preferably called

.../proddb??/human/assembly/first_run (for human), and
 .../proddb??/mouse/assembly/first_run (for mouse). If you don't have
 a local disk on this machine, make a directory called

\$BASEDIR/human/assembly//first_run (or use "mouse" for mouse),
 and cd to it.

now open a window on each machine you want to run the assembly, and do:

for human runs:

```
setenv RUN (usually run_name is the machine name, but for
            instance if you have both bioserv1 and bioserv2
            that both run on bioserv, the name should
            reflect which run you are doing).

setenv DIR $BASEDIR/human/cluster
setenv ORG human
setenv GBVER ??? (according to the version)
cp $DIR/$RUN.list .
assembly dir= $DIR in= rich_fasta.flipped out=embl.$RUN transc_file=transcr
```

for mouse runs:

```
setenv RUN (usually run_name is the machine name, but for
            instance if you have both bioserv1 and bioserv2
            that both run on bioserv, the name should
            reflect which run you are doing).

setenv DIR $BASEDIR/mouse/cluster
setenv ORG mouse
setenv GBVER ??? (according to the version)
cp $DIR/$RUN.list .
assembly dir= $DIR in= rich_fasta.flipped out=embl.$RUN transc_file=transcr
```

In case any of the runs ends abnormally, you have to start it again from
 the place it ended. To do this, you should look at the end of the "stdout"
 file of the run, and look for last appearance of something like this:

```
#####
###          begin          W70352.AA732187      (size =      15)      ###
#####
```

This means that the last contig the assembly tried to work on was called
 AA732187, in cluster W70352 (and it has 15 sequences in it). You should
 now make a new directory (for instance second_run), and rerun the assembly
 program, adding the parameter first=... (where ... is the last contig
 that was worked on - in this case W70352.AA732187).

If this run also fails on this contig, look in the \$RUN.list file, and
 see which contig comes after this one, and try starting the assembly run
 from this one.

IMPORTANT NOTE: The assembly appends it's output to the existing files,
 so if you started an assembly run, and it was stopped (either by itself
 or manually), if you want to rerun it, you should either do it in a
 different directory (recommended), or delete the existing embl, cview,
 keywords, transcripts, graph and stdout files.

It is very advisable to keep a list of the runs made in a file called
 Task.list at \$BASEDIR/human/assembly (or mouse), in the following format:

A-403

```
# computer    dir
bacio         /dir/bacio2/prodgb107/human/assembly/bacio/first_run
godiva        /dir/godiva2/prodgb107/human/assembly/godiva/first_run
mnm           /dir/bioserv2/prodgb107/human/assembly/mnm/first_run
lindt         /dir/lindt3/prodgb107/human/assembly/lindt/first_run
peru          /dir/peru1/prodgb107/human/assembly/peru/first_run
panama        /dir/panama1/prodgb107/human/assembly/panama/first_run
bioserv1      /dir/bioserv2/prodgb107/human/assembly/bioserv1/first_run
bioserv2      /dir/bioserv2/prodgb107/human/assembly/bioserv2/first_run
mir           /dir/bioserv2/prodgb107/human/assembly/mir/first_run
bacio         /dir/bacio2/prodgb107/human/assembly/bacio/second_run
```

29. MERGE:

We need now to merge all the results of the assembly runs into one file for each type. For this, we will use the script GuidedMerge. In the directory \$BASEDIR/human/assembly (or mouse for mouse), make a directory called final. Now we need to make a list of all the contigs we worked on - i.e. a merge of all of the .list files. For this do:

```
sort $BASEDIR/human/cluster/*.list >$BASEDIR/human/assembly/final/list.full
```

The command for the merge must be run under sh (and not csh or tcsh, because we need the stdout and stderr in different files), so the command is (for the embl file) is something like this:

```
sh -c 'GuidedMerge -embl $BASEDIR/human/assembly/final/list.full:list /dir/bacio
```

Because of limitations of interactive sh (only 256 characters on a line), and to avoid mistakes, there is a script that runs the command, based upon the Task.list file. The commands that should be run are:

```
run_merge_command.pl task_list= $BASEDIR/human/assembly/Task.list contig_list= $
run_merge_command.pl task_list= $BASEDIR/human/assembly/Task.list contig_list= $
run_merge_command.pl task_list= $BASEDIR/human/assembly/Task.list contig_list= $
run_merge_command.pl task_list= $BASEDIR/human/assembly/Task.list contig_list= $
run_merge_command.pl task_list= $BASEDIR/human/assembly/Task.list contig_list= $
```

You should check the .errs files of the merges - they should all be with the same number of lines, and the only kind of errors that should appear is lines like:

```
Couldn't find W70352.AA732187 (which means this contig wasn't found
                             in any of the output files)
```

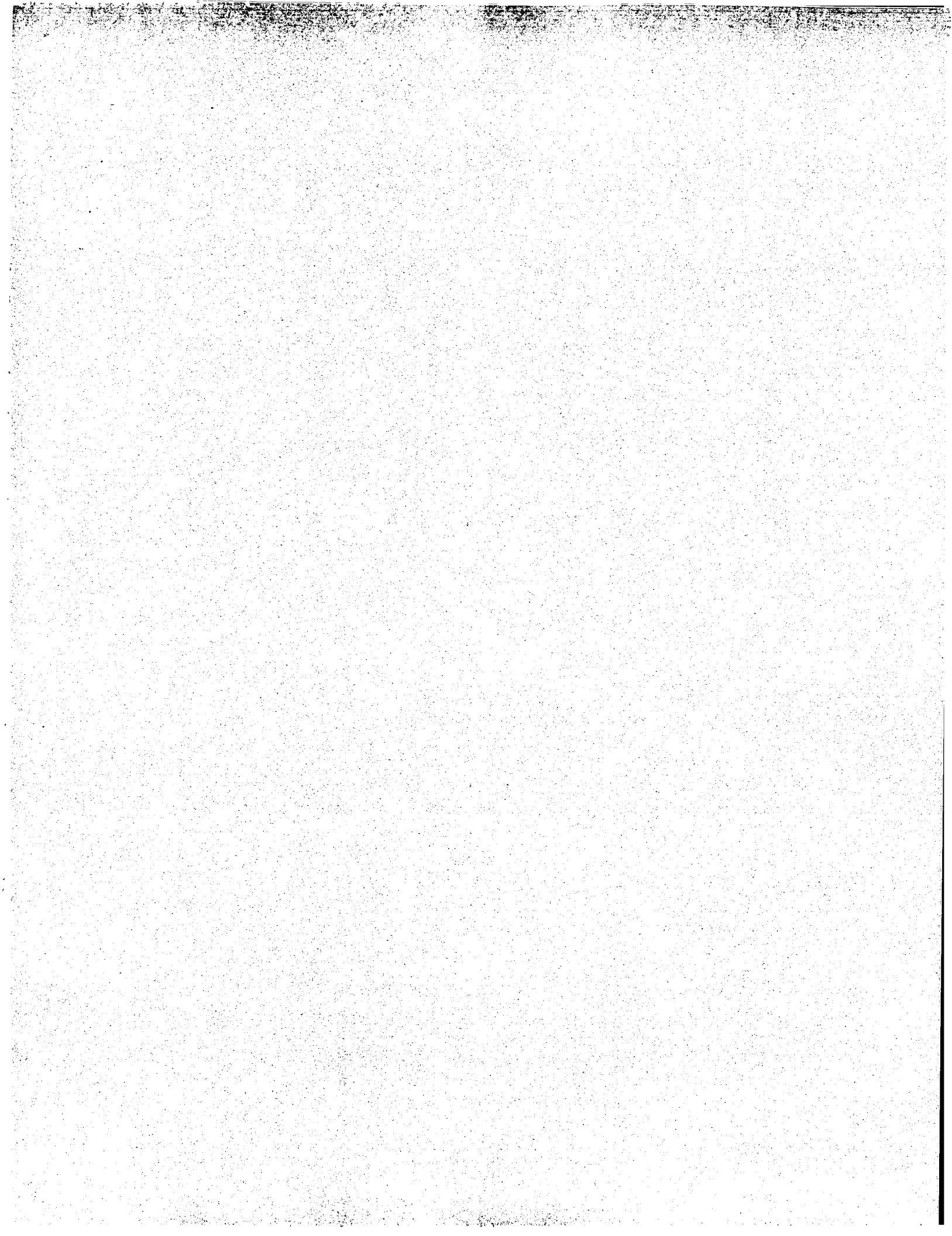
or:

```
Finished embl.bacio (which means this file was read until it's end).
```

30. HIERARCHY:

The cview file needs to be processed to make the input of the cluster viewer. To do this, you need to run the hierarchy program:

```
heirarchy dir= $BASEDIR/human/assmby/final in=cview.final out=cluster_view.fi
```



A-404

31. INSTALL DATABASE:

The following files should be moved to the database directories:

to the clusters directory:

1. rich_fasta.flipped (should be called there rich_fasta).
2. cluster_view
3. clusters.EMBL
4. graph

If any of the files (especially the last two in the human case) are larger than 2GB, they have to be split up into smaller files. This is done like this:

```
divide_file.pl in= $BASEDIR/human/assmbly/final/embl.final outprefix= $DATAB
```

```
divide_file.pl in= $BASEDIR/human/assmbly/final/graph.final outprefix= $DATA
```

29. MAKE_INDEX: some of the files that were produced are a part of the output of the project. These are the files:

1. rich_fasta.flipped
2. clusters.EMBL
3. transcripts
4. cluster_view

These files need to be indexed for quick access by GenCore utilities (important for GenWeb viewer). For this you only need a running version of GenCore, and then run "make_index". Each such run gives you another file (called "in_file.ind").

31. And that's all folks! You are now ready for the Dry biology phase!

Statistical Appendix

=====

In each stage of the clustering several numbers can be computed which give information about this particular version of clustering (and this particular database release). These numbers should be generated and formatted for display as a web page in Compugen's internal site. The pages from each version might vary, since the clustering flow varies, but here is a summary of the numbers calculated for version 105, and the way they were obtained:

raw data

```
est size      ls -l est
est seqs      grep "^>" est | wc -l
est bases     grep -v "^>" rna | wc (take third number minus first number)
rna size      ls -l rna
rna seqs      grep "^>" rna | wc -l
rna bases     grep -v "^>" rna | wc (take third number minus first number)
```


A-405

clean data

```

-----
est umsk size    ls -l est.umsk
rna umsk size    ls -l rna.umsk
est xout size    ls -l est.xout
rna xout size    ls -l rna.xout
cln ests         grep "^>" est.umsk | wc -l
cln rnas         grep "^>" rna.umsk | wc -l
cln est bases    grep -v "^>" est.umsk | wc   (third number - first number)
cln rna bases    grep -v "^>" rna.umsk | wc   (third number - first number)

```

without abundant genes

```

-----
est umsk size    ls -l est.umsk.ok
rna umsk size    ls -l rna.umsk.ok
est xout size    ls -l est.xout.ok
rna xout size    ls -l rna.xout.ok
cln ests         grep "^>" est.umsk.ok | wc -l
cln rnas         grep "^>" rna.umsk.ok | wc -l
cln est bases    grep -v "^>" est.umsk.ok | wc   (third number - first number)
cln rna bases    grep -v "^>" rna.umsk.ok | wc   (third number - first number)

```

first clustering

```

-----
total contigs    wc -l stat
singletons       uniq -c stat | tail (first number of last line)
doubletons       uniq -c stat | tail (first number of second line from bottom)
tripletons       uniq -c stat | tail (first number of third line from bottom)
quadrupletons    uniq -c stat | tail (first number of forth line from bottom)
quintupletons    uniq -c stat | tail (first number of fifth line from bottom)
biggest contig   head stat          (number in first line)
second contig    head stat          (number in second line)
third contig     head stat          (number in third line)
fourth contig    head stat          (number in fourth line)
fifth contig     head stat          (number in fifth line)
contigs > 500    awk '$1 > 500' stat | wc -l
contigs > 200    awk '$1 > 200' stat | wc -l

```

second clustering (for human only)

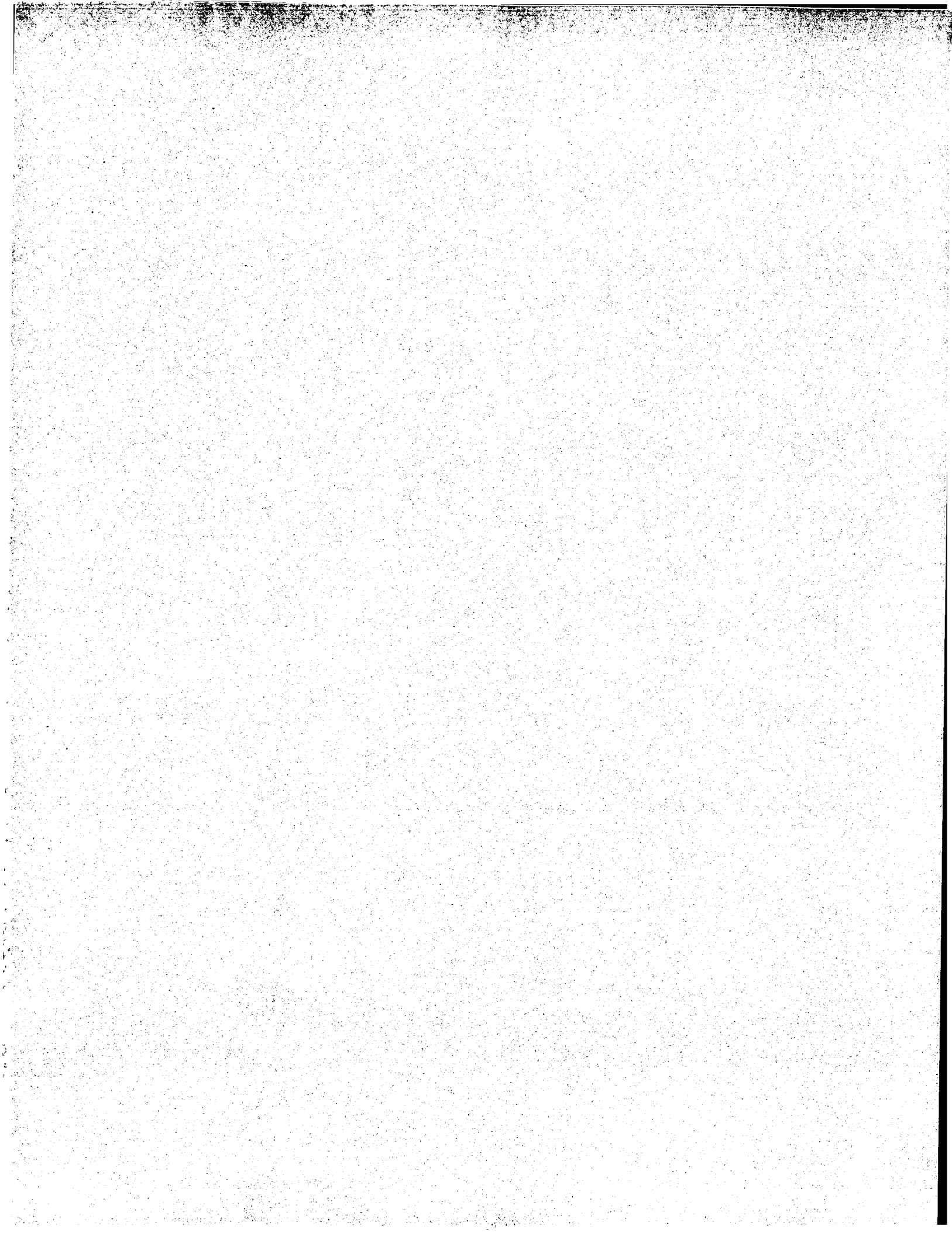
Same as first clustering - use the merged files

clone information

```

-----
clusters > 1     wc -l clone_stats
biggest cluster  head clone_stats (number in first line)
second biggest   head clone_stats (number in second line)
contig-triplets  uniq -c clone_stats | tail (first no., second line from bott
contig-pairs     uniq -c clone_stats | tail (first number bottom line)
standalone ctgs > 1  awk '$6 == 1 && $5 > 1' seq.clu | awk '(print $2)' | sort | u
standalone seqs   awk '$6 == 1 && $5 == 1' seq.clu | wc -l
avg contigs/cluster total_contigs / [(clusters>1)+(s.a. contigs>1)+(s.a. seqs)]
without single ctgs [total_contigs - (s.a. contigs>1) - (s.a. seqs)] / (clusters>
avg seqs/contig    (cln_ests+cln_rnas without abundant seqs) / total_contigs
without singletons (cln_ests+cln_rnas without abundant seqs - singletons) / (to

```



A-406

assembly

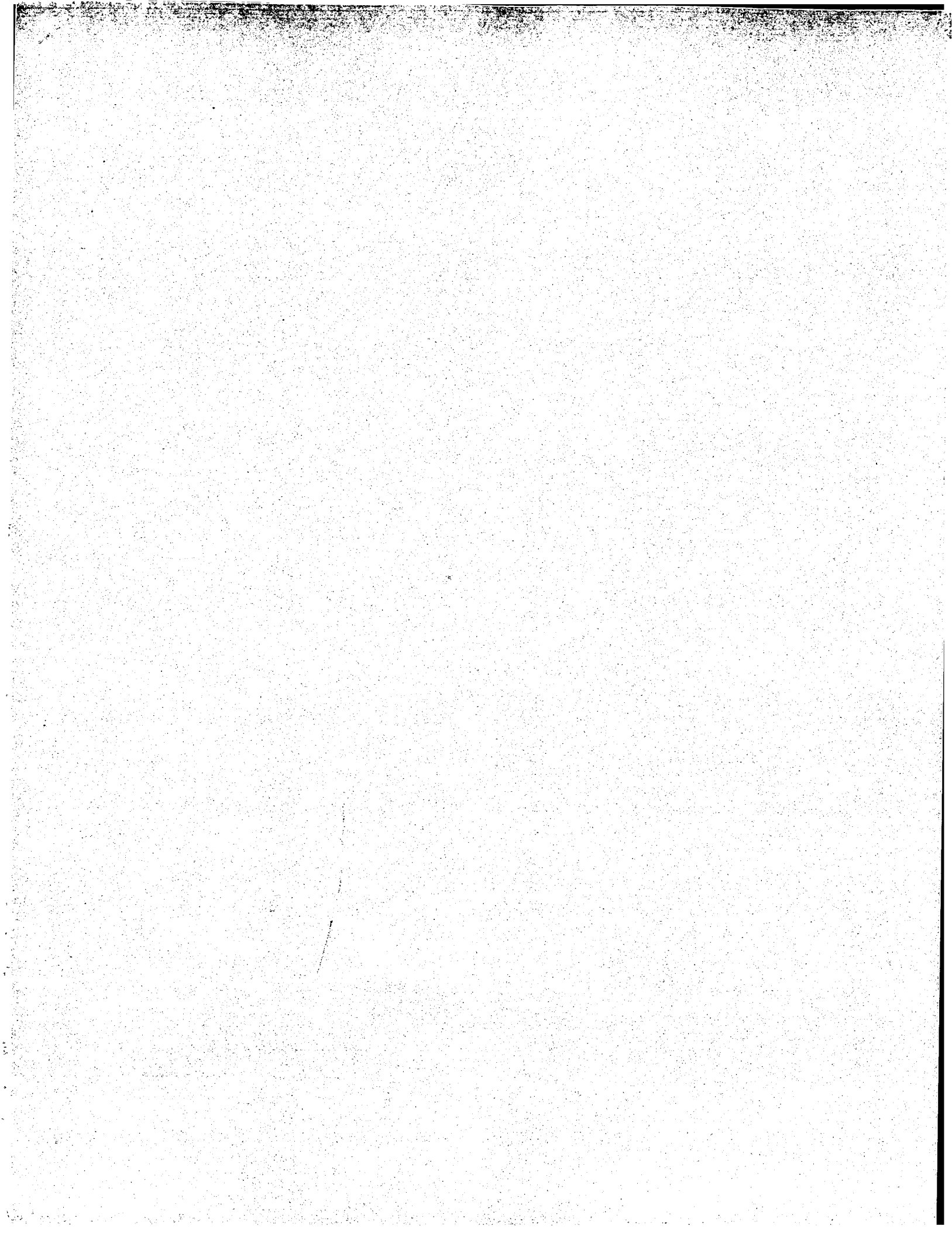
cluster avg len
without single ctgs
contig avg len
without singletons
singleton avg len
%alt splicing
Avg splice alts.
Where available

files sizes

obtained from "ls -l" on the following files:

est.gb
rna.gb
rich_fasta
transcripts
clusters.EMBL
keywords
cluster_view

total



[illegible][illegible][illegible]

26.

0827

10827

suppl. 2002-16163:231997

Listening for Sarah Pollock

SUN SEP 21 16:53:22 1997

```

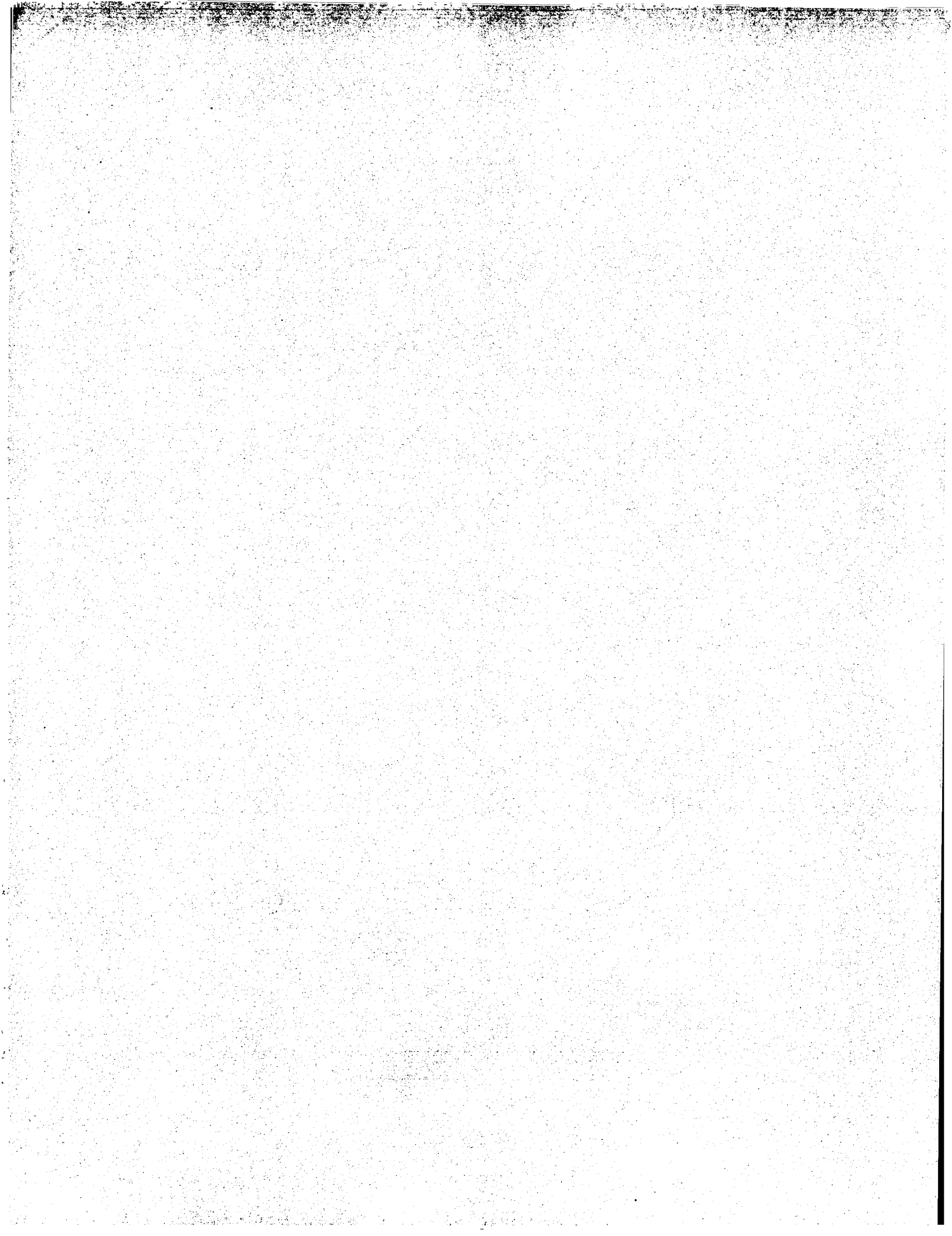
743 CTTTGGGCAAAACCTTCTCAAGGAAATCTCTAGCCACCGACAGAAAGTCGGT 792
1101 CTGTCAACCTGCATGATCTCTCCAGAGACACTTTCAGCCACGTCGGGTAGGGT 1150
793 CTGTCAACCTGCATGATCTCTCCAGAGACACTTTCAGCCACGTCGGGTAGGGT 842
1151 GTAGGGGGGAAGACAAAAGCAGCATGTGTGGAGCAGCTGGCTGACAAAGTT 1200
843 GTAGGGGGGAAGCAAAAGCAGCATGTGTGGAGCAGCTGGCTGACAAAGTT 892
1201 GCTGCSAAGCTGGCTCATACCTGTGTCCTCAATTAAGTTTCATGATCCGAAT 1249
893 GCTGCSAAGCTGGCTCATACCTGTGTCCTCAATTAAGTTTCATGATCCGAAT 941
to: CCM101_10827_1 from 1 to 1249
CCM101_10827 x CCM101_10827_1 ..
1 AGCAATATACAGGTAGCCCTGTAATAACATCCCTCCAGACTCTCACCT 50
1 AGCAATATACAGGTAGCCCTGTAATAACATCCCTCCAGACTCTCACCT 50
51 CTTTTTTATTCCTCCCAACCCCATGTTTGTGGATGATTAATTCACTAAAA 100
51 CTTTTTTATTCCTCCCAACCCCATGTTTGTGGATGATTAATTCACTAAAA 100
101 GAGAGAGAGACAGGTTTAATCCATATTGTGTCCTCCACCCACAGATT 150
101 GAGAGAGAGACAGGTTTAATCCATATTGTGTCCTCCACCCACAGATT 150
151 TTGGCTTTTCTCCCAAGGCTGATGACAAAGGCCCATGAATGATTTGA 200
151 TTGGCTTTTCTCCCAAGGCTGATGACAAAGGCCCATGAATGATTTGA 200
201 CTTAGCAACTCAATAGATCAGGGTTGGGCTCTACATTAAGAACAATTTT 250
201 CTTAGCAACTCAATAGATCAGGGTTGGGCTCTACATTAAGAACAATTTT 250
251 GACCTGGCTAGTCGGGACTATCATCCCTAACCATGATATAAGCAGACA 300
251 GACCTGGCTAGTCGGGACTATCATCCCTAACCATGATATAAGCAGACA 300
301 GGCTCTCTGTCTGGACATTTCTAAAGGTGAACAGTTGGGGGGGGGTGTT 350
301 GGCTCTCTGTCTGGACATTTCTAAAGGTGAACAGTTGGGGGGGGGTGTT 350
351 TCCTCTATTCACCTGTGTTTAAAGGAGGAGCAGAGGAGGTGACCTTA 400
401 CTCCTTTAATTCACCTGTGTTTAAAGGAGGAGCAGAGGAGGTGACCTTA 450
401 CTCCTTTAATTCACCTGTGTTTAAAGGAGGAGCAGAGGAGGTGACCTTA 450
451 CCCTAAGGTAAATGTGTCCTCTACCTGTCGTGTCAGTGTGTCACACTGCA 500
451 CCCTAAGGTAAATGTGTCCTCTACCTGTCGTGTCAGTGTGTCACACTGCA 500

```

2807

CC	501	TGCTGGTGCTGCTCATCATGCTGTGCATCATCTGCTACCAAGTGCATA	550
CC	501	TGCTGGTGCTGCTCATCATGCTGTGCATCATCTGCTACCAAGTGCATA	550
CC	551	TTTTCACTTCGGTGTCATCTCAGCTTTCCTTTTAAAAATTTCTCTCTCT	600
CC	551	TTTTCACTTCGGTGTCATCTCAGCTTTCCTTTTAAAAATTTCTCTCTCT	600
CC	601	TTTTTAACCTTTCTTAAATTAATTTTCTTTTAAAAAAGTTCACCAAGG	650
CC	601	TTTTTAACCTTTCTTAAATTAATTTTCTTTTAAAAAAGTTCACCAAGG	650
CC	651	AAACTAGTTCCCTCCACACAGCTCGGACAGGCTCCAGCCATCTCTTAAAA	700
CC	651	AAACTAGTTCCCTCCACACAGCTCGGACAGGCTCCAGCCATCTCTTAAAA	700
CC	701	ACCGCGCTGTTCTCCCTCACAATCCCAACAGGAGCTCTTTAGAGTGCTT	750
CC	701	ACCGCGCTGTTCTCCCTCACAATCCCAACAGGAGCTCTTTAGAGTGCTT	750
CC	751	TGGAGGGTGAAGTCTCTCCCTGTGTGGCGGATGGTCTGGGTAGTGAA	800
CC	751	TGGAGGGTGAAGTCTCTCCCTGTGTGGCGGATGGTCTGGGTAGTGAA	800
CC	801	GGCCCCCTGCCCACACAGAGGTTCTGATTTGTGGGACACAGTCTGGGTT	850
CC	801	GGCCCCCTGCCCACACAGAGGTTCTGATTTGTGGGACACAGTCTGGGTT	850
CC	851	GTTTCTCTCTCTCTCTTAAATAAAAAAAGACATTCCTGGATGACAGGCAC	900
CC	851	GTTTCTCTCTCTCTCTTAAATAAAAAAAGACATTCCTGGATGACAGGCAC	900
CC	901	AGTGACAGGAGTGAGCTTGAGCTATCGAAACACCTGGTGAAGTCTCTC	950
CC	901	AGTGACAGGAGTGAGCTTGAGCTATCGAAACACCTGGTGAAGTCTCTC	950
CC	951	AAGGCCACGACAACTTGCTTACATCTCTCGGCACATAGTGACTTGTCTGTG	1000
CC	951	AAGGCCACGACAACTTGCTTACATCTCTCGGCACATAGTGACTTGTCTGTG	1000
CC	1001	GAATCTGTAGTGTGTTTATGTGTCTACATGTGACAGCTCCACCTGTGGTCT	1050
CC	1001	GAATCTGTAGTGTGTTTATGTGTCTACATGTGACAGCTCCACCTGTGGTCT	1050
CC	1051	CTTTTGGCAAACTTTTCAAGGAATTTCTTAGCCACCGACAGAAAGTCGGT	1100
CC	1051	CTTTTGGCAAACTTTTCAAGGAATTTCTTAGCCACCGACAGAAAGTCGGT	1100
CC	1101	CTGTCAACCTGCATGATCTCCAGAGCATTAGCCACCGTCGGTAGGGT	1150
CC	1101	CTGTCAACCTGCATGATCTCCAGAGCATTAGCCACCGTCGGTAGGGT	1150
CC	1151	GTAGGGGGGAAGACAAAGCAGCATGTGTGGACGACGCTGGCTGACAGTTT	1200
CC	1151	GTAGGGGGGAAGACAAAGCAGCATGTGTGGACGACGCTGGCTGACAGTTT	1200
CC	1201	GCTGCSAAGCTGGCTCATCACCTGTGCCAATTTAGTTTCATGGATCCGAAT	1249
CC	1201	GCTGCSAAGCTGGCTCATCACCTGTGCCAATTTAGTTTCATGGATCCGAAT	1249

10827



Sun Sep 21 16:53:22 1997

to: AA238470 from 1 to 400

CCM101_10827 x AA238470

```
170 CTGTGATCAAAAGGCCCATCACTGAATTTGGACTTTAGCAACTCAATAGGAT 219
1 CTGTGATCAAAAGGCCCATCACTGAATTTGGACTTTAGCAACTCAATAGGAT 50
220 CAGGTTTGGGCTCTACATTTAAAGCAAACTTTGACCTTGCTAGTCCGGAC 269
51 CAGGTTTGGGCTCTACATTTAAAGCAAACTTTGACCTTGCTAGTCCGGAC 100
270 TATCATCCCTTAACCATGATATAAGCAGACGAGCTCCTTTGTCTGGACATT 319
101 TATCATCCCTTAACCATGATATAAGCAGACGAGCTCCTTTGTCTGGACATT 150
320 TCTAAAGGTGAACAGATTGGGGGGGGGTGTTTCCCTCTATTCACTGTGTT 369
151 TCTAA-----155
370 TAAAGGAGGAGAGCAGAGGAGGTGACCTTAACTTTTAACTTCCAACTGT 419
156-----155
420 AACTATGTTACCTACTCGCATCCCGCGCTTCCCTAAGGATAATGTGTCTG 469
156-----155
470 TCTACTCTGCTGTCTCAGTGTGTGCACTGTCATGCTGGTGTGTCTATCA 519
156-----155
520 TGTGTGTGCATCATCTGCTACCAGGTGCATATTTCAGTTCTGGGTGCATC 569
156-----155
570 TCAGCTTTCTCTTTTAAAAAATTTTCTTTCTTTTAACTTTTCTTAAAT 619
156-----155
620 TATTTTTCCTTTTAAAGAGTGTCAACAGGAACCTAGTTCCTCCCTCCAC 669
::-----AGAAAGAGATTCACCAAGGAACCTAGTTCCTCCCTCCAC 192
670 CAGCTCGGACAGGCTCCAGCCATCTTTAAAAACCAGCGCTGTTTCTCCCTC 719
193 CAGCTCGGACAGGCTCCAGCCATCTTTAAAAACGCGCTGTTTCTCCCTC 242
720 ACATCCCAACAGAGGACCTCCTTAAGATGCTTGGAGGGTGAAGTCTCTCC 769
243 ACATCCCAACAGAGGACCTCCTTAAGATGCTTGGAGGGTGAAGTCTCTCC 292
770 CTGTGTGGGGGATGCTGTGTGTGTGTGTGTGTGTGTGTGTGTGTGTGTGT 819
293 CTGTGTGGGGGATGCTGTGTGTGTGTGTGTGTGTGTGTGTGTGTGTGTGT 342
820 AGGTTTCTGTGTGTGGGACACAGTCTGGGTTGTTTTTCTCTCTCTCTCTA 869
343 AGGTTTCTGTGTGTGGGACACAGTCTGGGTTGTTTTTCTCTCTCTCTCTA 392
```

```

870 ATAAAAA 877
|||||
393 ATAAAAA 400
|||||

to: AA189833 from 1 to 362
CCM101_10827 x AA189833 rev

744 GATGCCCTTGAGGGTGAAGTCTCTCCCTGTGTGGCGGGATGCTCTGGTGG 793
|||||
1 GATGCCCTTGAGGGTGAAGTCTCTCCCTGTGTGTGGCGGGATGCTCTGGTGG 50
|||||
794 TAGTGAAGGCCCTCTGCCACAACAGAGGTTTCTGATTCTGGGACACAGT 843
|||||
51 TAGTGAAGGCCCTCTGCCACAACAGAGGTTTCTGATTGTGGGACACAGT 100
|||||
844 CTGGGTTGTTTTCTTCTCTGTCTTCTTAATAAAAAAAGACATTCTCGATGA 893
|||||
101 CTGGGTTGTTTTCTTCTGTCTTCTTAATAAAAAAAGACATTCTCGATGA 150
|||||
894 CAGGCACAGTGCAGGAGTGTGAGCTTTGAGCTATCGAAACAACCTCGGTGAA 943
|||||
151 CAGGCACAGTGCAGGAGTGTGAGCTTTGAGCTATCGAAACAACCTCGGTGAA 200
|||||
944 GTCTCTCAAGGCCACGAAACCTTTGCTTTACATTCTCTCGGCACCTAGTGCATT 993
|||||
201 GTCTCTCAAGGCCACGAAACCTTTGCTTTACATTCTCTCGGCACCTAGTGCATT 250
|||||
994 GCTTGTGGAAAATCTGTAAAGTTGTTTATGTGTCACTGTGCACAGCTCCCACT 1043
|||||
251 GCTTGTGGAAAATCTGTAAAGTTGTTTATGTGTCACTGTGCACAGCTCCCACT 300
|||||
1044 GTGCTCTCTTTTTGGCAAACTTTTCAAGGAAATTTCTTAGCCACCGACAGAA 1093
|||||
301 GTGCTCTCTTTTTGGCAAACTTTTCAAGGAAATTTCTTAGCCACCGACAGAA 350
|||||
1094 AGTCGGTCTGTCTC 1105
|||||
351 AGTCGGTCTGTCTC 362

to: AA215106 from 1 to 434
CCM101_10827 x AA215106 rev

60 CCCTTCCCAACCCCATGTTTGTGGATGAATAATTCATTAAGAAGAGAGAG 109
|||||
1 CCCTTCCCAACCCCATGTTTGTGGATGAATAATTCATTAAGAAGAGAGAG 50
|||||
110 AGCAGGTTTAATCCATATTTGTCTCTCCACCCACAGTTTGGCTTTT 159
|||||
51 AGCAGGTTTAATCCATATTTGTCTCTCCACCCACAGTTTGGCTTTT 100
|||||
160 CTCGCCAAGGCTGTATGACAAAGGCCCATGAATCGATTGGACTTTAGCAAC 209
|||||
101 CTCGCCAAGGCTGTATGACAAAGGCCCATGAATCGATTGGACTTTAGCAAC 150
|||||
210 TCAATAGGATCAGGGTGGGCTCTACATTAATAAGCAACCTTTGACCTGGC 259
|||||

```

10827

10827

Sun Sep 21 16:53:23 1997

Listing for Sarah Pollock

```
CC CC
151 TCAATAGGATCAGGGTGGGCTCTACATTAAGAGCACTTACCTGGC 200
CC CC
260 TAGTCCGGACTATCACTCCCTAACCATATATAGCAGACAGGCTCCCTTG 309
CC CC
201 TAGTCCGGACTATCACTCCCTAACCATATATAGCAGACAGGCTCCCTTG 250
CC CC
310 TCTGCACTTTCTAAAGGTGAACAGTTGGGGGGGGGTGTTCCCTCTATT 359
CC CC
251 TCTGCACTTTCTAAAGGTGAACAGTTGGGGGGGGGTGTTCCCTCTATT 300
CC CC
360 CACCTGTGTTTAAAGGAAGAGAGAGAGAGAGGTGACCTACCTCTTTAA 409
CC CC
301 CACCTGTGTTTAAAGGAAGAGAGAGAGAGGTGACCTACCTCTTTAA 350
CC CC
410 TTCCAACTGTAACTATGTACTACTCGCATCCCGCGCTTCCCTAAGGT 459
CC CC
351 TTCCAACTGTAACTATGTACTACTCGCATCCCGCGCTTCCCTAAGGT 400
CC CC
460 AATGTGCTGTC-TACTGTGCTGCTCAGTGT 492
CC CC
401 AATGTGCTGTCATCTGCTGCTCAGTGT 434
CC CC
to: AA285612 from 1 to 230
CCM101_10827 x AA285612 rev
CC CC
230 CTCTACATTAAGCAACTTTGACCTGGCTAGTCCGGACTATCATCCCT 279
CC CC
1 CTCTACATTAAGCAACTTTGACCTGGCTAGTCCGGACTATCATCCCT 50
CC CC
280 AACCATGATATAAGCAGACCAAGGCTCCTGTCTGGACATTTCTAAAGTG 329
CC CC
51 AACCATGATATAAGCAGACCAAGGCTCCTGTCTGGACATTTCTAAAGTG 100
CC CC
330 AACAGTT-GGGGGGGGGTGTTCCTCTATTACCTGTGTTTAAAGGAG 378
CC CC
101 AACAGTTGGGGGGGGGGTGTTCCTCTATTACCTGTGTTTAAAGGAG 150
CC CC
379 GAGACAGAGAGAGGTGACCTTACCTCTTTAAATCCAACTGTAACTATGTT 428
CC CC
151 GAGACAGAGAGAGGTGACCTTACCTCTTTAAATCCAACTGTAACTATGTT 200
CC CC
429 ACCTACTCGCATPCCCGCGCTTCCCTAAGG 458
CC CC
201 ACCTACTCGCATPCCCGCGCTTCCCTAAGG 230
CC CC
to: AA137379 from 1 to 504
CCM101_10827 x AA137379 rev
CC CC
744 GATGCTTGGAGGGTGAAGTCTCTCCCTGTGGGGGAGTGGTCTGGTG 793
CC CC
1 GATGCTTGGAGGGTGAAGTCTCTCCCTGTGGGGGAGTGGTCTGGTG 50
CC CC
794 TAGTGAAGCCCTCTCCCAACAGAGAGTGTCTGATTGTGGGACACAGT 843
CC CC
51 TAGTGAAGCCCTCTCTCCCAACAGAGAGTGTCTGATTGTGGGACACAGT 100
CC CC
```

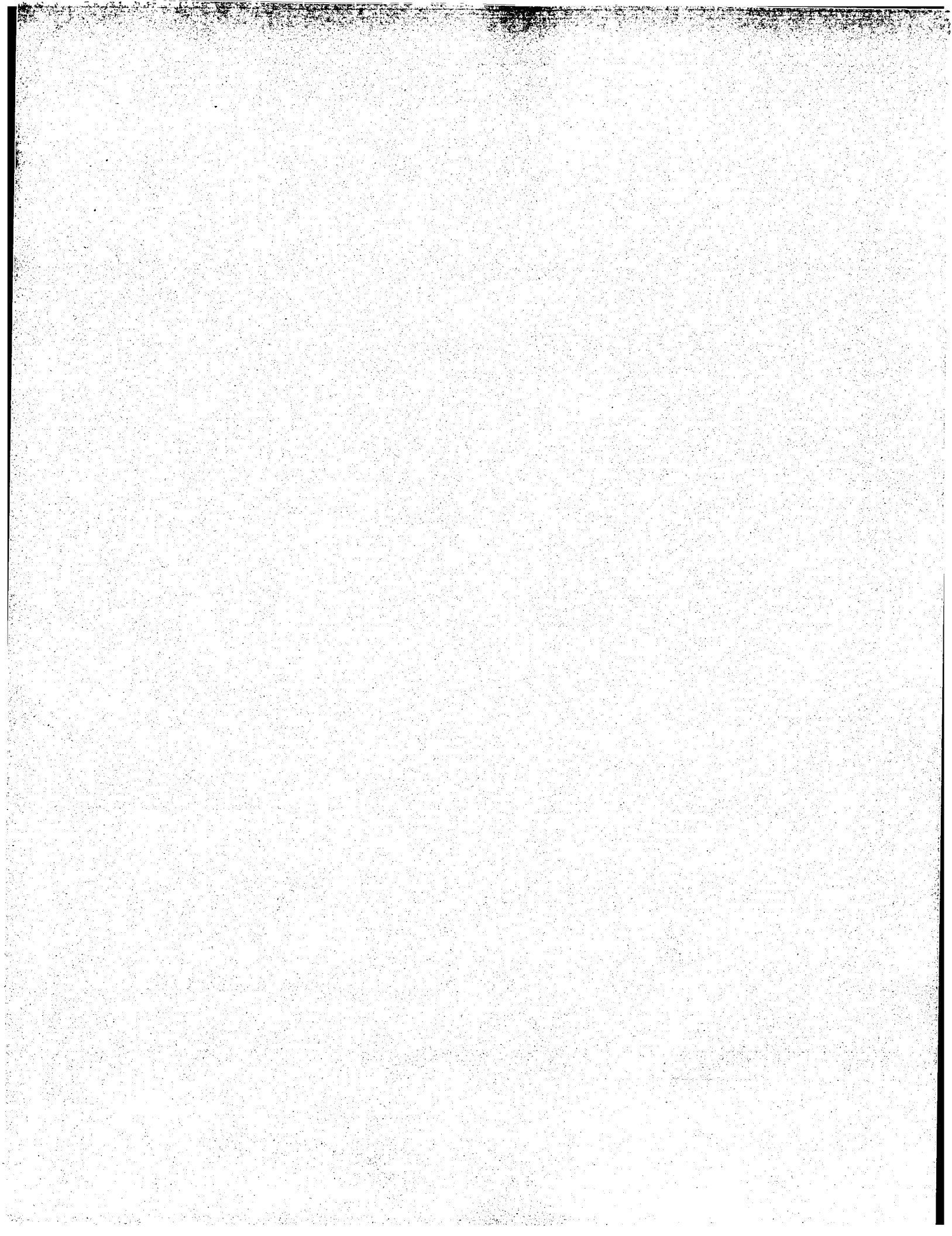
10827

Sun Sep 21 16:53:23 1997

Listing for Sarah Pollock

```
CC CC
844 CTGGGTTGTTTCTTCCCTGTCTCTAAATAAAAAAGACATTTCTCGATGA 893
CC CC
101 CTGGGTTGTTTCTTCCCTGTCTCTAAAT-AAAAAAGACATTTCTCGATGA 149
CC CC
894 CAGGCACAGTCAGGAGTGTGAGCTTGTGAGCTATCGAAACAACCTGGTGA 943
CC CC
150 CAGGCACAGTCAGGAGTGTGAGCTTGTGAGCTATCGAAACAACCTGGTGA 199
CC CC
944 GTCTCTCAAGGCCAGCAAACTTGTCTTACATTTCTCGGCACTAGTGACTT 993
CC CC
200 GTCTCTCAAGGCCAGCAAACTTGTCTTACATTTCTCGGCACTAGTGACTT 249
CC CC
994 GCTTGTGAAATCTGTAAAGTTGTTTATGTCTCACTGTGACAGCTCCACT 1043
CC CC
250 GCTTGTGAAATCTGTAAAGTTGTTTATGTCTCACTGTGACAGCTCCACT 299
CC CC
1044 GTGGTCTCTTTGGCAAACTTTTCAGGAATTTCTTAGCCACCGACAGAA 1093
CC CC
300 GTGGTCTCTTTGGCAAACTTTTCAGGAATTTCTTAGCCACCGACAGAA 349
CC CC
1094 AGTCGGTCTGTCAACCTGTCATGATCTCCAGAGACACTTCAGCCACCTCGG 1143
CC CC
350 AGTCGGTCTGTCAACCTGTCATGATCTCCAGAGACACTTCAGCCACCTCGG 399
CC CC
1144 GTAGGTTGTAGGGGGAAGACAAAGCAGCATGTGTGAGCAGCTGGCTG 1193
CC CC
400 GTAGGTTGTAGGGGGAAGACAAAGCAGCATGTGTGAGCAGCTGGCTG 449
CC CC
1194 ACAAGTTCTCSCAAGCTGGCTCATCATCTGTCCAAATTTAGTTTCATGAT 1243
CC CC
450 ACAAGTTCTG-GCAAAGCTGGCTCATCATCTGTCCAAATTTAGTTTCATGAT 498
CC CC
1244 CCGAAT 1249
CC CC
499 CCGAAT 504
CC CC
to: AA255271 from 1 to 395
CCM101_10827 x AA255271 rev
CC CC
683 CTCCAGCATCTTTAAAAACCGCGTGTTCCTCTCACTCCCAAAACAG 732
CC CC
1 CTCCAGCATCTTTAAAAACCGCGTGTTCCTCTCACTCCCAAAACAG 50
CC CC
733 GGAATCTCTTAGGATGCTTGGAGGGTGAAGTCTCTCCCTGTGGCGGA 782
CC CC
51 GGAATCTCTTAGGATGCTTGGAGGGTGAAGTCTCTCCCTGTGGCGGA 100
CC CC
783 TGGTCTGGTGTAGTGAAGGCCCTCTGCAACAAGAGGTTCTGATTG 832
CC CC
101 TGGTCTGGTGTAGTGAAGGCCCTCTGCAACAAGAGGTTCTGATTG 150
CC CC
833 TGGGACACAGTCTGGGTGTTTCTTCTCTGCTCTTAATAAAAAAGACA 882
CC CC
151 TGGGACACAGTCTGGGTGTTTCTTCTCTGCTCTTAATAAAAAAGACA 200
CC CC
883 TTCTTGGATGACAGGACAGTCAGGAGTGTGAGCTTGTGAGCTATCGAAG 932
CC CC
```

10827



Listing for Sarah Pollock
Sun Sep 21 16:53:23 1997

```

CC 1033 AGCTCCCACTGTGGTCTCTTTTGGCAACCTTTTCAAGGAATTTCTCTAGCC 108
CC 201 AGCTCCCACTGTGGTCTCTTTTGGCAACCTTTTCAAGGAATTTCTCTAGCC 250
CC
CC 1083 ACCGACAGAAAGTCGGTCTGTCTCAACCTGCATGATCTCCACAGACACTTCA 1132
CC
CC 251 ACCGACAGAAAGTCGGTCTGTCTCAACCTGCATGATCTCCACAGACACTTCA 300
CC
CC 1133 GCCACGTCCGGTAGGTGTAGGGGGGAAGACAAAGACAGCATGTGTGGAG 1182
CC
CC 301 GCCACGTCCGGTAGGTGTAGGGGGGAAGACAAAGACAGCATGTGTGGAG 350
CC
CC 1183 CAGCTGGCTGACAAAGTTGCTGCSAAGCTGGCTCATCACCTGTGCCAATTA 1232
CC
CC 351 CAGCTGGCTGACAAAGTTGCTGGCCAGAGCTGGCTCATCACCTGTGCCAATTA 400
CC
CC
CC
to: AA209012 from 1 to 411
CCM101_10827 x AA209012
CC
CC 158 WTCCTCCCA-AGGCTGATGACAAAGGCCCATGAACCTGGATTGGACTTAGC 206
CC
CC 1 AT-TCGGCAGAGGCTGATGACAAAGGCCCATGAACCTGGATTGGACTTAGC 49
CC
CC 207 AACTCAATAGGATCAGGGTTGGGCTCTACATTAATAAGCAACTTTGACCCCT 256
CC
CC 50 AACTCAATAGGATCAGGGTTGGGCTCTACATTAATAAGCAACTTTGACCCCT 99
CC
CC 257 GGCTAGTCGGACTATCATTCCTTAACCATGATATAAGCAGACCAAGGCTCC 306
CC
CC 100 GGCTAGTCGGACTATCATTCCTTAACCATGATATAAGCAGACCAAGGCTCC 149
CC
CC 307 TTGCTCTGCACATTTCTAAAGTGAAACAGTTGGGGGGGGTGTTCCTCTCT 356
CC
CC 150 TTGCTCTGCACATTTCTAA----- 167
CC
CC 357 ATTACCTCTGTGTTTTAAAGGAAGAGCAGACGAGAGGTGACCCCTACCTCTT 406
CC
CC 168 ----- 167
CC
CC 407 TAAATCCAACTGTAACCTATGTTACCTACTCGCATCCCGCGGTTCCTTAA 456
CC
CC 168 ----- 167
CC
CC 457 GGTAATGTGCTGTCTACTGTCTGTCTCAGTGCTGCACATGCATGCTGG 506
CC
CC 168 ----- 167
CC
CC 507 TGTGTGCTATCATGTGTGTGCATCATCTGCTAACGATGATATTTTCAG 556
CC
CC 168 ----- 167
CC
CC 557 TTCTGGGTGTCATCTCAGCTTTCCTTTTAAAAATTTTCTTCTTTTTTTA 606
CC
CC 168 ----- 167
CC
CC 607 ACTTTCTTTAAATTAATTTTTCCTTTTAAAGAGATTTACACAGAGAACTA 656
CC
CC 168 -----AGAAAGAGTTTACACAGAGAACTA 191

```

70827

Sun Sep 21 16:53:23 1997

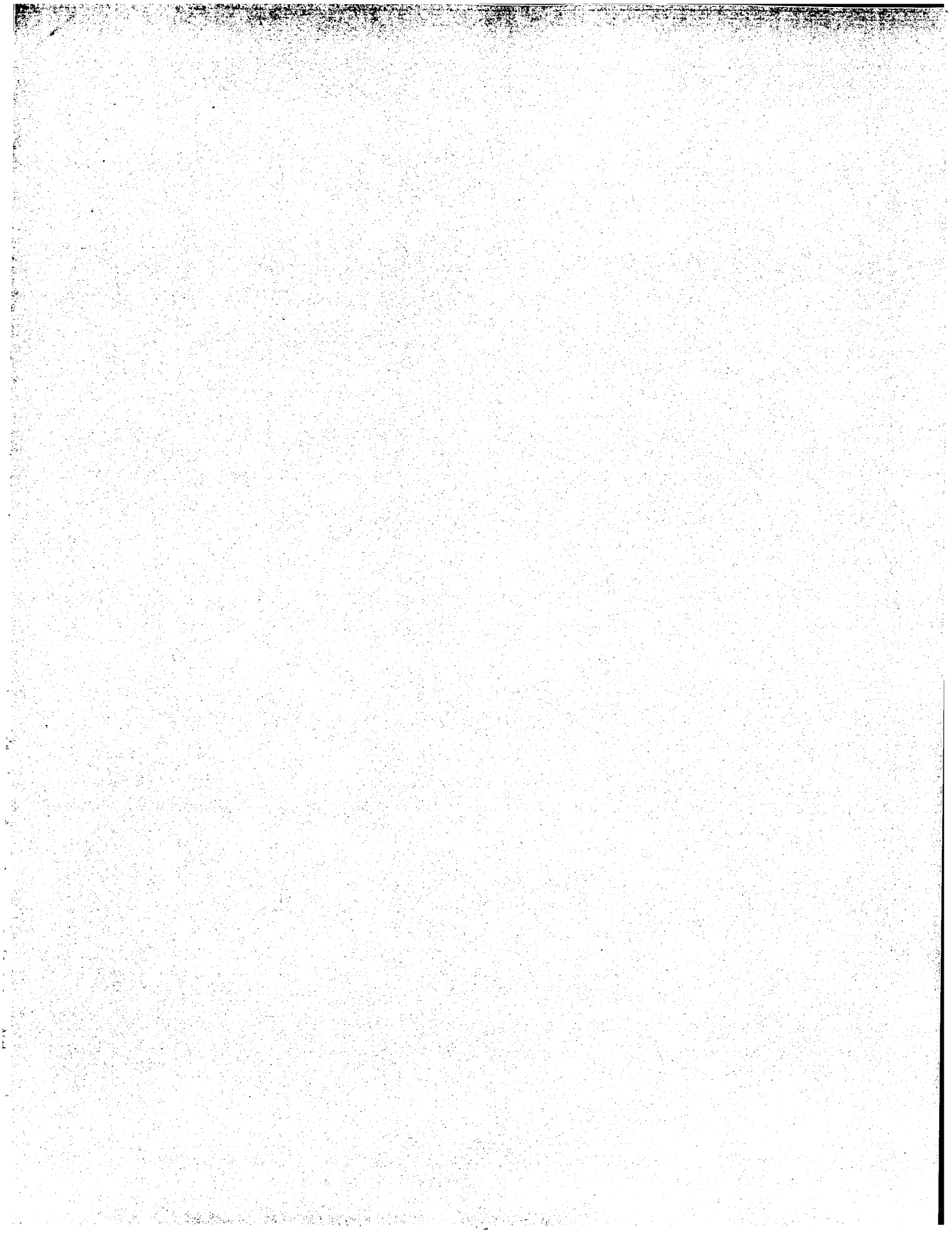
```

401 CCTCTTTAAATTCACAACGTAAACATATGCTTACCCTACTCCGCAAT 440
to: AA415881 from 1 to 273
CCM101_10827 x AA415881 rev ..
444 CGCGCTTCCTAAGGTAATGTCGTCTCT-ACTGTGCTGTCTCAGTGTGT 492
1 CGCGCTTCCTAAGGTAATGTCGTCTCAGTGTCTCAGTGTGTCTCAGTGTGT 50
493 GCACTGCATCGTGGTGTGTCCTATCATGTGTGTGCATCATCTGTGTACCA 542
51 GCACTGCATCGTGGTGTGTCCTATCATGTGTGTGCATCATCTGTGTACCA 100
543 GGTGCATATTTCAGTTCTGGTGTCTCATCTCAGCTTTCTCTTTTAAAAATTT 592
101 GGTGCATATTTCAGTTCTGGTGTCTCATCTCAGCTTTCTCTTTTAAAAATTT 150
593 TCTTTCTCTTTTAACTTTTCTTAAATTAATTTTCTCTTTTAAAGAAGT 642
151 TCTTTCTCTTTTAACTTTTCTTAAATTAATTTTCTCTTTTAAAGAAGT 200
643 TCACAGGAATACTAGTTCCCTCCACACAGCTCGGACAGGCTCCAGCCAT 692
201 TCACAGGAATACTAGTTCCCTCCACACAGCTCGGACAGGCTCCAGCCAT 250
693 CCTTAAACACGCGCTGTTCTC 715
251 CCTTAAACACGCGCTGTTCTC 273
to: AA415392 from 1 to 380
CCM101_10827 x AA415392 rev ...
336 TGGGGGGGGGTGTTTCCTCTATTACCTGTGTTTAAAGGAGGAGAGCA 385
1 TGGGGGGGGGTGTTTCCTCTATTACCTGTGTTTAAAGGACGGAGAGCA 50
386 GAGGAGGTGACCCCTACCTCTTTAATTCCACTGTAACTATGTTACCTACT 435
51 GAGGAGGTGACCCCTACCTCTGTAATTCCACTGTAACTATGTTACCTACT 100
436 CGCATCCCGCGCTTCCTTAAGGTAATGTCGTCT-CTACTGTCGTGCT 484
101 CGCATCCCGCGCTTCCTTAAGGTAATGTCGTCTGCTACTGTCGTGCT 150
485 CAGTGTGTGCACTGCATGCTGGTGTGTCTATCATGTGTGTGCATCATC 534
151 CAGTGTGTGCACTGCATGCTGGTGTGTCTATCATGTGTGTGCATCATC 200
535 TGTACACAGGTGCATATTTTCAGTTCGTGGTGTCTCTCAGCTTTCCCTTTT 584
201 TGTACACAGGTGCATATTTTCAGTTCGTGGTGTCTCTCAGCTTTCCCTTTT 250
585 AAAAATTTTCTCTTTTAACTTTTCTTAAATTAATTTTCTCTTTTAG 634
251 AAAAATTTTCTCTTTTAACTTTTCTTAAATTAATTTTCTCTTTTAG 300

```

10827

10827



Listing for Sarah Pollock

Sun Sep 21 16:53:23 1997

gatggtctgg tggtagtga ggcctctctg ccacacaga ggtttctgat tgtgggacac 840
agctgggtt gttttcttc tgtttctaa taaaaaaga cattctctga tgacaggcac 900
agtgcaggag tgtgagcttg agctatcgaa acaacttgtt gaagtctctc agggcccgac 960
aaacttgtt acattctctg gcactagtga ctgtctgtg gaaactctga agttgtttat 1020
gctcacctgt gacagctccc actgtggtctt ctttggcaa acccttcaag gaattctcta 1080
gccaccgaca gaaagtcggt ctgtcaacct gcatgatctc ccagagcact tcagccacgt 1140
cgggtagggt gtaggggga agacaaaagc agcatgtgtg gaggagctgg ctgacaaagt 1200
gctgcscag ctggctcacc accgtgtcaa ttagtctatg gatccgaat 1249

//

Listing for Sarah Pollock

Sun Sep 21 16:53:23 1997

635 AAAGAAGTTCACGAGAACTAGTTCCCTCCACACACGTCGGACAGGCT 684
|||||
301 AAAGAAGTTCACGAGAACTAGTTCCCTTAACACACGTCGGACA-GCT 349
685 CCAGCCATCTTAAAA-ACCGGCTGTTTCT 714
|||||
350 CCAGCCATCTTAAATATACCGGCTGTTTCT 380
|||||

to: AA450899 from 1 to 374

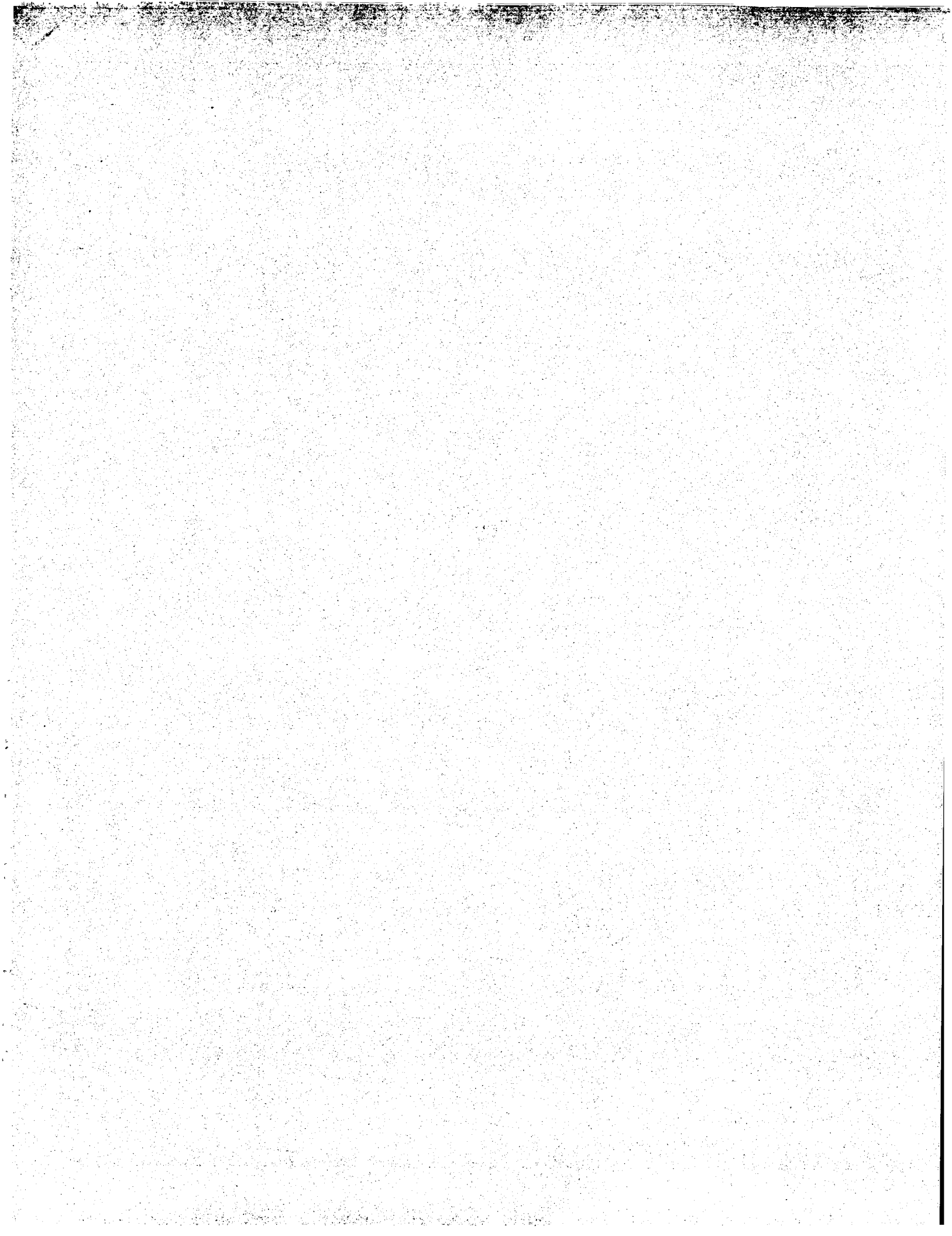
CCM101_10827 x AA450899 rev ..

861 TGTCTTCTTAATAAAAAAGACATTCCTGGATGACAGGCACAGTCGAGGAG 910
1 TGTCTTCTTAATAAAAAAGACATTCCTGGATGACAGGCACAGTCGAGGAG 50
911 TGTGAGCTTGAGCTATCGAANAACCTGGTGAAGTCTCTCAAGGCCAGC 960
51 TGTGAGCTTGAGCTATCGAANAACCTGGTGAAGTCTCTCAAGGCCAGC 100
961 AAACCTTGCTTACATTCCTCGGCACCTAGTACTTGTGGAAATCTGTA 1010
101 AAACCTTGCTTACATTCCTCGGCACCTAGTACTTGTGGAAATCTGTA 150
1011 AGTTGTTTATGTGTCACCTGTCAGACGTCCTCCACCTGTTGTTGGCAA 1060
151 AGTTGTTTATGTGTCACCTGTCAGACGTCCTCCACCTGTTGTTGGCAA 200
1061 ACCTTTCAAGGAATTTCTTAGCCACCGACAGAAAGTCGCTGTCACCT 1110
201 ACCTTTCAAGGAATTTCTTAGCCACCGACAGAAAGTCGCTGTCACCT 250
1111 GCATGATCTCCAGAGCACTTCAGCCACCTCGGGTAGGGTCTAGGGGGA 1160
251 GCATGATCTCCAGAGCACTTCAGCCACCTCGGGTAGGGTCTAGGGGGA 300
1161 AGACAAAGCAGCATGTGTGAGCAGCTGGCTGACAAAGTTCCTGCSAAG 1210
301 AGACAAAGCAGCATGTGTGAGCAGCTGGCTGACAAAGTTCCTGCTCC-CAAG 349
1211 CTGGCTCATCACTGTCCAAATTAGT 1235
350 CTGGCTCATCACTGTCCAAATTAGT 374
|||||

PFS end.
Sequence 1249 BP; 294 A; 309 C; 281 G; 363 T; 2 other:
agcaataac aggtagccct gtaatacat tccctccaga cctccacct ctttttatc 60
cttccacc ccattgttgt ggaataataa ttcactaaaa gagagagaga gcaagttta 120
atccatatt gtccctcca cccaccagtt ttggtctwlc tcccaagc tgatgacaaa 180
ggccatgaa ctgattgga cttagcaact caataggatc aggttgggc tctacattaa 240
agcaactt gacctgggt agtccgact acatoccta accatgatat aagcagacca 300
ggctcttgt ctggaacttt ctgaaggaga aggttgggg 99999999tt tctctatcc 360
acctgttt aaaggaaga gagcagaga ggtgacctta cctctttaa tccactgta 420
actatgtac ctactgcac ccccgctt cctaaagta atgtgtctgt ctactgtct 480
gtctcagtg gtgcactga tctgtgtgtg tctatcat gtgtgtgat catctgtac 540
caggcgata tttagtgtt ggtgtcatc cagcttctct ttttaaaa ttcttctt 600
ttttaact tttaaaatt attttctt ttgaagaa gtccacagg aaactgttc 660
cctccacc acgtgggaca ggtccagcc atcctaaa accgcgtgt tctccctca 720
catcccaac agggactct taggatgct tggagggtga agtctctccc tgtgtgggg 780

10827

10827



B-8

Li Ming (or Sarah) Pollock

Sim, Sep 21/16/53/28/1997

>CCM101_10827_0 Mus musculus (mouse) CCM101_10827
agcaataacaggtgagccctgtgaaataacataatccctccagagactctccctctctttttatc
ctctccaccacatgtttgtggatgaataatcactcaaaagagagagagcaaggttta
atccatattctctccaccacagttttggcttwtctcccaaggtctgacaaa
ggcccatgaactgaggttggacttagcaactcaataggatcaggttgggtctacattaa
aagcaactttgacccctgtgctccggactatccctcaaccatgatatagcagacca
ggctctgttctggacatttctaaagtgaaagcttgggggggggtttctctctattc
acctgtttaaagagagagagagagagagaggtgacctaccctcttaattccaaactga
actagttaactaactgcaccccggttccctaaagtaagtgtctctactactgtctg
gtctcagtgctgacagctgctggtggtgctcactatcgtgctgacatctctgctac
caggtgcataattcagttctgtgctcagctctctcttttaaaaaattctctctt
ttttaactttcttaaatattttctctcttaagaagaagttcacccaggaactagttc
cctcccccacgtcggaaggtctcagccatccttaaaacccgctgtttctccctca
catcccaacagggaccccttggatgcttgggggtgaagctctcctctgttggggg
gatgtctgtgtgaggtgagggccctctgcacaaagaggtttctgttggggacac
agctcgggttgttttctctctctcttaataaaaaagacattctgtgacagggcac
agtcagagaggtgagcttgaactatcgaacaacactggtgaagctctcaagggccagc
aaactgttctactctctcctggactaggttctgtggaatctgtgaagttgtttat
gtgtcactgtgacagctccactgtggtctcttttggcaaccccttcaaggaattctcta
gccaccagaaagagctgtctcaactgacatgctccagagcacttcagccagt
cgggtaggtgtgagggggaagacaaagcagcagctgtgtgagcagcttgctgacaagt
gatgcgaagctgctcactcactctcaattagttcattggtccgaat
>CCM101_10827_1 Mus musculus (mouse) CCM101_10827
agcaataacaggtgagccctgtgaaataacataatccctccagagactctccctctttttatc
ctctccaccacatgtttgtggatgaataatcactcaaaagagagagagcaaggttta
atccatattctctccaccacagttttggcttwtctcccaaggtctgacaaa
ggcccatgaactgaggttggacttagcaactcaataggatcaggttgggtctacattaa
aagcaactttgacccctgtgctccggactatccctcaaccatgatatagcagacca
ggctctgttctggacatttctaaagaagaagttccaccaggaactagttccctccca
ccaagtcagaggtccagccatcctcaaaacccgctgtttctccctccatcccaa
acagggactcctttaggactcttggaggtgaagctctcctctgttgggggaggtgct
gggtgagtgaaagggccctctgcacaaagacattctgttgggacacagcttggg
ttgtttctctctctcttaataaaaaagacattctgttgggacacagcttggg
agtgtgagcttgaagctatgaacaacactgtgaagctctcaagccagcaacttgc
ttacattctcggactaggtgcttcttgggaaactctgaagttgtttatgtgtaact
gtgacagctccactgtgctctcttttgggaaaccttcaaggaattctctagccaccga
cagaaagctggtctgtcaacctgcatgactccagagcacttcagccacgtcgggtagg
gtgtagggggggaagacaaagcagcagctgtgtggagcagctgctgacaagtgtgtgcsa
agctggtcctacaccctgtcccaattagttcattggtccgaat

10827.transcript

851 GCCACCTGCGCTCCCTCCCAACCTCTGATGAAACTGCACGAGAGAAAGCT 900
901 CACAGCAAGCTGTGATTCATGTCCTCCCATCCCGCTCTCACAGCCAAAT 950
901 CACAGCAAGCTGTGATTCATGTCCTCCCATCCCGCTCTCACAGCCAAAT 950
951 GCACACAAAGTCTTCCTGGCTGAGGAGCAGAGAGAAAGTAAACAATGGA 1000
951 GCACACAAAGTCTTCCTGGCTGAGGAGCAGAGAGAAAGTAAACAATGGA 1000
1001 GCTTGTGAAAGAAAGACGACAGCTGTGCGGTGTGATGTTTCCCGATGTC 1050
1001 GCTTGTGAAAGAAAGACGACAGCTGTGCGGTGTGATGTTTCCCGATGTC 1050
1051 TTCACTTGTGTCCTCCATGTTGGAGCCCTCCCTCTGCTGTTCCCAAA 1100
1051 TTCACTTGTGTCCTCCATGTTGGAGCCCTCCCTCTGCTGTTCCCAAA 1100
1101 GTCCATGGATCCGAAT 1116
1101 GTCCATGGATCCGAAT 1116
to: R75296 from 1 to 314
CCM101_15537 x R75296 ..
1 AGACTTTAAGTAAGCAATTTATTTATTTCCCTGGCTGACAGATTGA 50
1 AGACTTTAAGTAAGCAATTTATTTATTTCCCTGGCTGACAGATTGA 50
51 CTTCAAGTACAGAACTGAGCTGACAAACAGACGATGATGCTC 100
51 CTTCAAGTACAGAACTGAGCTGACAAACAGACGATGATGCTC 100
101 CCAACAGGAGGAGCGCTCTTCATTTGGTGTAGATGAAATGTCATGT 150
101 CCAACAGGAGGAGCGCTCTTCATTTGGTGTAGATGAAATGTCATGT 150
151 GAGATGCAATGCTGCGCTCTCCCTCCCAACAGAGTCCCTAAATGAAATGA 200
151 GAGATGCAATGCTGCGCTCTCCCTCCCAACAGAGTCCCTAAATGAAATGA 200
201 TGAGTAAAGCCCTCATGAACTCTGCGGCCCTCTCACTCCATCTCTTT 250
201 TGAGTAAAGCCCTCATGAACTCTGCGGCCCTCTCACTCCATCTCTTT 250
251 GAATAGCAGATGCCAGGAGGAGAGCAGTGGCCATGCTTTAGCATTTGC 300
251 GAATAGCAGATGCCAGGAGGAGAGCAGTGGCCATGCTTTAGCATTTGC 300
301 AGTGGTGTGAGAC 314
301 AGTGGTGTGAGAC 314
to: AA200163 from 1 to 263
CCM101_15537 x AA200163 rev ..

842 TCTTAGCGGGCCACTGCGCTCTCCCAACCTCTGATGAAACTGCAAGCA 891
1 TCTTAGCGGGCCACTGCGCTCTCCCAACCTCTGATGAAACTGCAAGCA 50
892 GAGAAAGCTCACAGAAAGCTGTGATTCATGTCGTCGCCATCCCGCTCTC 941
51 GAGAAAGCTCACAGAAAGCTGTGATTCATGTCGTCGCCATCCCGCTCTC 100
942 ACAGCCCAATGCAACACAAAGTCTTCCTGGCTGAGGAGAGCAAGAAAGT 991
101 ACAGCCCAATGCAACACAAAGTCTTCCTGGCTGAGGAGAGCAAGAAAGT 150
992 AACAAATGAGCTTGTGAAAGAAAGACGACGCTGTGCGGTGTGATGTTT 1041
151 AACAAATGAGCTTGTGAAAGAAAGACGACGCTGTGCGGTGTGATGTTT 200
1042 CCCGATGCTTTCACCTTGGCTCTCTCCATGTTGGAGCCCTCCCTCTG 1091
201 CCCGATGCTTTCACCTTGGCTCTCTCCATGTTGGAGCCCTCCCTCTG 249
1092 GTTCCCAAGTCC- 1104
250 GTTCCCAAGTCC 263
to: W75922 from 1 to 405
CCM101_15537 x W75922 rev ..
561 TAGAGAGAGGAGGATGAGGAGACAGAGGTTAGGGGTACAAATGCTTCAAGTA 710
1 TAGAGAGAGGAGGATGAGGAGACAGAGGTTAGGGGTACAAATGCTTCAAGTA 50
711 CAGTGGGTATCTTTTGGTCCCTTAGGTGTGCT-CCCTACGGCATTTCTTTT 759
51 CAGTGGGTATCTTTTGGTCCCTTAGGTGTGCTCCCTACGGCATTTCTTTT 100
760 CCCCTTT-TGCCGCTGCGCTTTGTACAAATGCCAAATCAAACTCTCTGC 808
101 CCCCTTTATGCGCTGCGCTTTGTACAAATGCCAAATCAAACTCTCTGC 150
809 CCAAGAGAGCGG--AGTTGCTGTAAATTTGATTTCTTCTAGCGGCCACC 856
151 CCAAGAGAGCGGCGAGTTGCTGTAAATTTGATTTCTTCTAGCGGCCACT 200
857 TGCCCTCTCCCAACCTCTGATGAACTGACAGAGAGAAAGTCAACAGC 906
201 TGCCCTCTCCCAACCTCTGATGAACTGACAGAGAGAAAGTCAACAGC 250
907 AAAGCTGTGATTCATGTCGTCTCCCATCCCGCTCTCACAGCCAAATGCAACA 956
251 AAAGCTGTGATTCATGTCGTCTCCCATCCCGCTCTCACAGCCAAATGCAACA 300
957 CAAAGTCTTCTGGGCTGAGGAGCAGAGAAAGTAAACAATTTGAGCTTGT 1006
301 CAAAGTCTTCTGGGCTGAGGAGCAGAGAAAGTAAACAATTTGAGCTTGT 350
1007 GAAAAAGAGCAGAGCTGTGCGGTGTGATGTTTCCCGATGCTCTTCAACC 1056

Sun Sep 21:17:13:21 1997

[illegible]

CC	636	TGAAGCCTGGGTGGAAGAAAAGCACTAGGAGAGGAAGGTATGGAGACAGA	688
CC	CC		
CC	401	TGAAGCCTGGGTGGAAGAAAAGCACTAGGAGAGGAAGGTATGGAGACAGA	450
CC	CC		
CC	686	GGTTAGGGGTACAAATGCTTCAGTGAC	711
CC	CC		
CC	451	GGTTAGGGGTACAAATGCTTCAGTGAC	476
CC	CC		
CC	to: AA242484 from 1 to 290		
CC	CCM101_15537 x AA242484 rev		
CC	CC		
CC	557	CACGGCCCTATTTTCCAAGAGGACTCTAAACGTCCTAAACGTCCTGTTCGAA	606
CC	CC		
CC	1	CACGGCCCTATTTTCCAAGAGGACTCTAAACGTCCTAAACGTCCTGTTCGAA	50
CC	CC		
CC	607	TGTCCTCTCCGCCACAACTGGGGCGGAGCTATGAAGCTCGGGTGGAAAGAAAA	656
CC	CC		
CC	51	TGTCCTCTCCGCCACAACTGGGGCGGAGCTATGAAGCTCGGGTGGAAAGAAAA	100
CC	CC		
CC	657	GGACTAGGAGGAAGAGGTATGGAGACAGAGGTTAGGGGTACAATGCTTCA	706
CC	CC		
CC	101	GGACTAGGAGGAAGAGGTATGGAGACAGAGGTTAGGGGTACAATGCTTCA	150
CC	CC		
CC	707	GTGACAGTGGGTATCTTTTGGTCCCTTAGGTGTGT-CCCTACGGCATTC	755
CC	CC		
CC	151	GTGACAGTGGGTATCTTTTGGTCCCTTAGGTGTGTCCCTACGGCATTC	200
CC	CC		
CC	756	TTTTTCCCTTTGCCCGTCGGCTGTACAAATGCCAAAAATCAAACCTC	805
CC	CC		
CC	201	TTTTTCCCTTTGCCCGTCGGCTGTACAAATGCCAAAAATCAAACCTC	250
CC	CC		
CC	806	TGCCCAAAGAGCGG--AGTTGCTGCTTAAATTTGATTCTTC	843
CC	CC		
CC	251	TGCCCAAAGAGCGGCGAGTTGCTGCTTAAATTTGATTCTTC	290
CC	CC		
CC	to: W81846 from 1 to 303		
CC	CCM101_15537 x W81846 rev		
CC	CC		
CC	17	CAATATTTTATTTTCCCTGGCTGACAGCATTGACTTCAAAAGTACCAGAA	66
CC	CC		
CC	1	CAATATTTTATTTTCCCTGGCTGACAGCATTGACTTCAAAAGTACCAGAA	50
CC	CC		
CC	67	CCTGAGCTGCACAAACAGAGCTACAGATTGCTCCCAACAGGGAGGACGC	116
CC	CC		
CC	51	CCTGAGCTGCACAAACAGAGCTACAGATTGCTCCCAACAGGGAGGACGC	100
CC	CC		
CC	117	TCCTTCAATGGTAGAGATGAAATTGAAATGTCATGTGAGAAATGCAATGCTG	166
CC	CC		
CC	101	TCCTTCAATGGTAGAGATGAAATGAAATGTCATGTGAGAAATGCAATGCTG	150
CC	CC		
CC	167	CCCCCTCCCCCACAAGAGTCCCTAAATGAAATGAAATGAGTAAAGCCCCCAT	216
CC	CC		
CC	151	CCCCCTCCCCCACAAGAGTCCCTAAATGAAATGAAATGAGTAAAGCCCCCAT	200
CC	CC		
CC	217	GAAACTCTGGGGCCCTCTCACTCCATCTTCTTTGAAATAGCAGATGCCAC	266
CC	CC		

15537

15537

150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968

201 AGTAAAGCCCCCATGAACCTCTGGGCCCCCTCTCACTCCATCTTCTTGA 250

253 ATAGCAGATGCCACGGAGGGAGACAGCTGGCCATCGCTTAGCATTTGCAG 302

[illegible]

452 GGCTCTGAAAGTCCAAAGGTACTTATGGGATCGAGGACGACATGTCCA 501
450 GGCTCTGAAAGTCCAAAGGTACTTATGGGATGAGGACGACGACATGTCCA 499
502 ACCGTGAATCCCACTGTTTAATGGCCATCCAAAGCGTGGCAATCATAAACCG-- 550

A238329, from 1 to 435
1_15537 x A238329 rev ..
1171 TCCTCCCCACAGATCCCTAAATGAATGAATGAGTAAAGCCCCCATGAAA 220

1 TCCCCCCACAGAGTCCCTAAATGAAATGAATGAGTAAGCCCCCATGAAA 50

2221 CTCTGGGGCCCCCTCTCACTCCATCTTTTGAATAGCAGATGCCACGGAG 270

51 CTCTGGGGCCCCCTCTCACTCCATCTTCTTTGAATAGCAGATGAAACGGAG 100

271 GGAGACAGCTGGCCATCGCTTAGCATTTGCAGTGCCTGTGAGACCCAAGA 320

221 GACTCAGCAGGAGAGACGAGTGGCGAAGCTGGCCATCTTACCT 370
151 GACTGCAGCAGGAGAGACGAGTGGCGAAGCTGGCCATCTTACCT 200
171 TGTTAACCGGACCCTCCACACACAGACGCTGCCCTCTACATGAGAA 420

15537

15537


```

449 CCCCAAGTCCATGATCCGAAT 470
to: AA269487 from 1 to 309
CCM101_15537 x AA269487 rev ..
290 TTATGATTTGCGTGTGTGAGACCCAGAGACTGCAGCAGGGAGAGGA 339
1 TTATGATTTGCGTGTGTGAGACCCAGAGACTGCAGCAGGGAGAGGA 50
340 CGAGTGGGCAAGTGGGCAACCTAGCTTACCTTGTAAACCGGACCCCTCCAC 389
51 CGAGTGGGCAAGTGGGCAACCTAGCTTACCTTGTAAACCGGACCCCTCCAC 400
390 ACACAGACACGCTGCCCTGTACATGAGAAAGGTGAAGGAAATGCTAGAG 439
101 ACACAGACACGCTGCCCTGTACATGAGAAAGGTGAAGGAAATGCTAGAG 489
440 GACTGATCCTGAGGCTCTCAAAGTCCAAGGTACTTTATGGGATGAGAGC 539
151 GACTGATCCTGAGGCTCTGAAAGTCCAAGGTACTTTATGGGATGAGAGC 589
490 AGGACATGTTCAACCCCTGAACCTCCAGTTTAATGTCATCCAAAACGCTGGC 639
201 AGGACATGTTCAACCCCTGAACCTCCAGTTTAATGTCATCCAAAACGCTGGC 689
540 AATCATAAACGTTCCGCGACGCGCTATTTTCCAAGAGGACTCTAAACGT 739
251 AATCATAAACGTTCCGCGACGCGCTATTTTCCAAGAGGACTCTAAACGT 789
590 CTAACGCTC 598
301 CTAACGCTC 309
to: W18243 from 1 to 352
CCM101_15537 x W18243 rev ..
229 CCCCTCTCACTCCATCTCTTTTGAATAGCAGATGCCACGAGGAGACAG 278
1 CCCCTCTCACTCCATCTCTTATGATAGCAGATGCCACGAGGAGACAG 328
279 CTGCGCCATCGCTTAGCATTTGCGAGTGGGTGTGAGACCCAGAGACTGCG 378
51 CTGCGCCATCGCTTAGCATTTGCGAGTGGGTGTGAGACCCAGAGACTGCG 428
329 CAGGAGAGGACGAGTGGGCAAGTGGGCAACCTAGCTTACCTTGTAAACCC 478
101 CAGGAGAGGACGAGTGGGCAAGTGGGCAACCTAGCTTACCTTGTAAACCC 528
379 GGACCCCTCCACACACAGACACGCTGCCCTGTACATGAGAAAGGTGAAGG 578
151 GGACCCCTCCACACACAGACACGCTGCCCTGTACATGAGAAAGGTGAAGG 628
429 AAATGCTAGAGGACTGATCTCTGAGGCTCTCTGAAAGTCCAAGGTACTTATG 678
201 AAATGCTAGAGGACTGATCTCTGAGGCTCTCTGAAAGTCCAAGGTACTTATG 728

```


SUN SEP 21 17:13:21 1997

Sun Sep 21 17:13:21 1997

Listing for Sarah Pollock

```

CC 479 GGATGAGGAGCAGGACATGTTCTCAACCTGAACTCCAGTTTAAATGCCATCC 529
CC 251 GGATGAGGAGCAGGACATGTTCTCAACCTGAACTCCAGTTTAAATGCCATCC 300
CC
CC 529 AAAACGCTGGCAATCATATAACCGTTCCGACAGCGCCCTATTTTCCAAAGAG 578
CC 301 AAAACGCTGGCAATCATATAACCGTTCCGACAGCGCCCTATTTTCCAAAGAG 350
CC
CC 579 AC 580
CC 351 AC 352
CC
CC to: AA250280 from 1 to 436
CC CCM101_15537 x AA250280 rev
CC
CC 290 TTATGATTTTGACGTGCTGTGAGACCCAAAGAGACTGCGACGAGGAGAGA 339
CC 1 TTATGATTTTGACGTGCTGTGAGACCCAAAGAGACTGCGACGAGGAGAGA 50
CC 340 CGAGTGGGCAAGTGGGCAACCTAGCTTACCTTGTTAAACCGGACCCCTCCAC 389
CC 51 CGAGTGGGCAAGTGGGCAACCTAGCTTACCTTGTTAAACCGGACCCCTCCAC 100
CC 390 ACACAGACACGCTGCCCTCTGTACATGAGAAAGGTGAAGGAAATGCTAGAG 439
CC 101 ACACAGACACGCTGCCCTCTGTACATGAGAAAGGTGAAGGAAATGCTAGAG 150
CC 440 GACTGATCCTGAGGCTCCTGAAAGTCCAAAGTCACTTATGGGATGAGGAGC 489
CC 151 GACTGATCCTGAGGCTCCTGAAAGTCCAAAGTCACTTATGGGATGAGGAGC 200
CC 490 AGGACATGTTTCAACCCCTGAACTCCAGTTTAAATGCGCATCCAAAACGCTGCG 539
CC 201 AGGACATGTTTCAACCCCTGAACTCCAGTTTAAATGCGCATCCAAAACGCTGCG 250
CC 540 AATCATAACCGTTCCCGCAGCGCCCTATTTTCCAAAGAGACTCTATAACGT 589
CC 251 AATCATAACCGTTCCCGCAGCGCCCTATTTTCCAAAGAGACTCTATAACGT 300
CC 590 CTAAACGCTCTGTTGCAATGCTCTCCGCCACAACTGGGGCGGAGCTATGAA 639
CC 301 CTAAACGCTCTGTTGCAATGCTCTCCGCCACAACTGGGGCGGAGCTATGAA 350
CC 640 GCCTGGCTGCAAGAAAAGGACTAGGAGAGGAAGGTATGGAGACAGAGGTT 689
CC 351 GCCTGGGTGGAAGAAAGGACTAGGAGAGGAAGGTATGGAGACAGAGGTT 400
CC 690 AGGGGTACAATGCTTCCAGTGACAGTGGGTATCTTTTT 725
CC 401 AGGGGTACAATGCTTCCAGTGACAGTGGGTATCTTTTT 436
CC
CC to: AA396216 from 1 to 438
CC CCM101_15537 x AA396216 rev
CC
CC 664 GAGAGGAAGGTATGGAGACAGAGTTTAGGGGTACAATGCTTCAGTGACAG 713
CC

```

15597

1 GAGAGGAAGGATATGGAGACAGAGGTTAGGGGGTACAAATGCTTCAGTGCAG 50
 714 TGGGTATCTTTTGGTCCCTTAGTGTGGTCCCT-ACGGCATTTCTTTTCCC 762
 51 TGGGTATCTTTTGGTCCCTTAGTGTGGTCCCTTACGGCATTTCTTTTCCC 100
 763 CTTTTCGGCTGGCTGTGTAACAATGCCAAAAATCAAACTCTGCCCAA 812
 101 CCTTTGCCGTGGCTGTGTAACAATGCCAAAAATCAAACTCTGCCCAA 150
 813 AGAGCCG---AGTTGCTGTAAATTTGTAATTTCTTAGCGGCCACTGCC 860
 151 AGAGCCGCGAGTTGCTGTAATTTGTAATTTCTTAGCGGCCACTGCC 200
 861 CTCTCTCCAAACCTCTGTATGAACCTGCACAGAGAAAGCTACACGAAG 910
 201 CTCTCTCCAAACCTCTGTATGAACCTGCACAGAGAAAGCTACACGAAG 250
 911 CTGTGATTCATCTGCTGCCCATCCCGCTCTCACAGCCAAATGCAACACAAG 960
 251 CTGTGATTCATCTGCTGCCCATCCCGCTCTCACAGCCAAATGCAACACAAG 300
 961 TTCTTCTCGGCTGAGGAAGCAGAGAAAAGTAACAATTTGAGCTTGTGAAA 1010
 301 TTCTTCTCGGCTGAGGAAGCAGAGAAAAGTAACAATTTGAGCTTGTGAAA 350
 1011 AAAAGACGACGCGTGGGTGTGATGTTTCCGATGCTCTCACCTTGC 1060
 351 AAAAGACGACGCGTGGGTGTGATGTTTCCGATGCTCTCACCTTGC 400
 1061 GTCTCTCCATCGTTGGAGCGCTCCCTCTGTTTCCCAAGT 1102
 401 GTCTCTCCATCGTTGGAGCGCTC---ATGGAT-CCGAAT 438
 to: W20607 from 1 to 347
 CCNM101_15537 x W20607 rev
 385 TCACACACAGACGCTGCGCCCTGTACATGAGAAAGGTGAAGGAATGC 434
 1 TCACACACACAGACGCTGCGCCCTGTACATGAGAAAGGTGAAGGAATGC 50
 435 TAGAGGACTGATCTCTGAGGCTCTGAAAGCTCCAAAGTACTTATGGGATGA 484
 51 TAGAGGACTGATCTCTGAGGCTCTGAAAGCTCCAAAGTACTTATGGGATGA 100
 485 GGAGCAGGACATGTTTCAACCTGAACTCCAGTTTAAATGCCATCCAAACG 534
 101 GGAGCAGGACATGTTTCAACCTGAACTCCAGTTTAAATGCCATCCAAACG 150
 535 CTGGCAATC---ATAACCGTTCCGCGACGCGCTATTTTCCAGAGGACTC 582
 151 CTGGCAATCTTTTAAACCGTTCCGCGACGCGCTATTTTCCAGAGGACTC 200
 583 TAAACGCTTAAACGCTGTGTAATGCTCTCCGCAACAACCTGGGGCGGAG 632
 201 TAAACGCTTAAACGCTGTGTAATGCTCTCCGCAACAACCTGGGGCGGAG 250
 633 CTATGAAGCCTTGGGTGGAAGAAAAGACTAGAGAGGAAGGTATGGAGAC 682

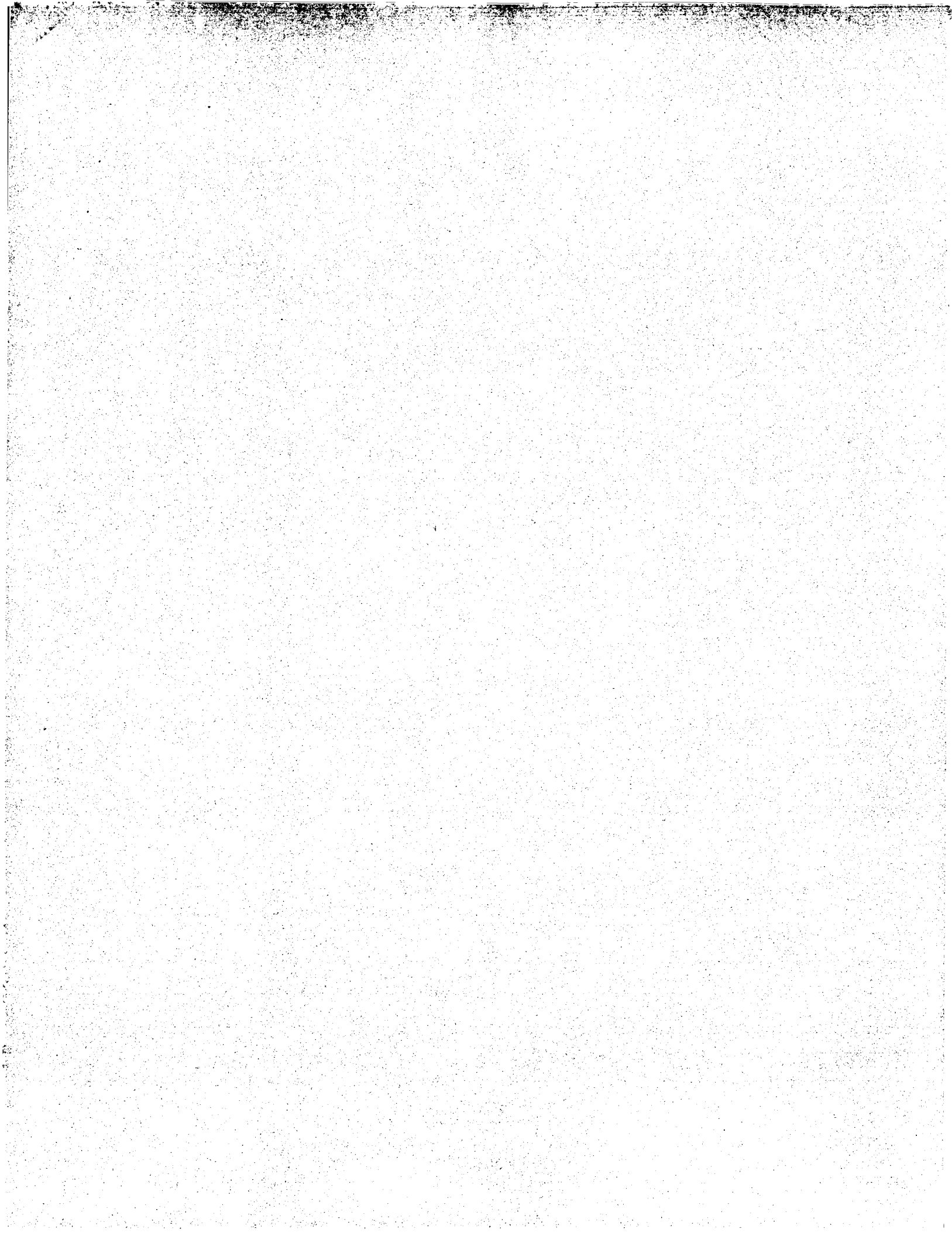
15537

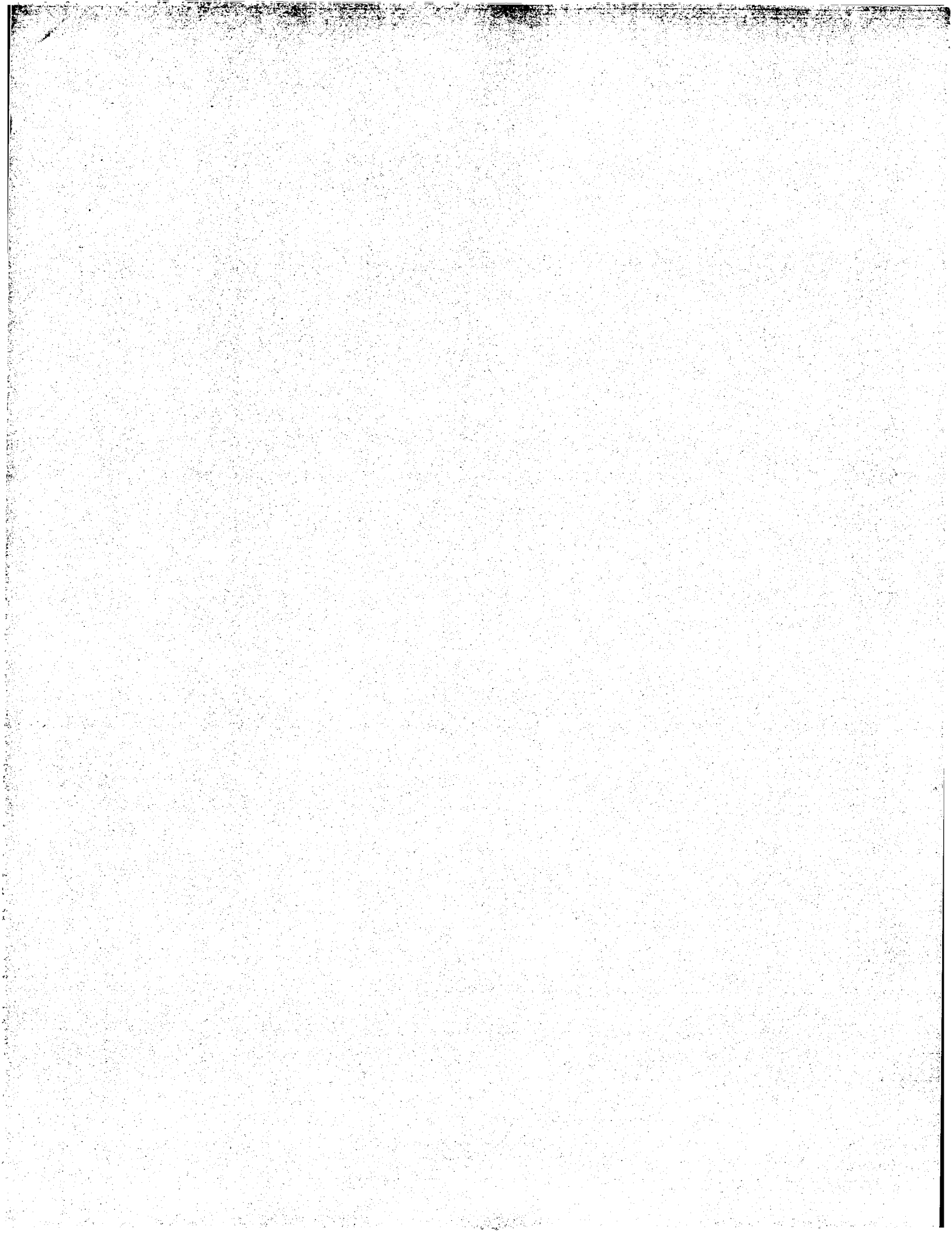
6-14

Sun Sep 21 17:13:21:1997

265 ACAGGAGGAGACAGCTGGCCATCGCTTGTAGCATTTGCGAGTGCCTGCGTGTGAGAC 314
 101 ACAGGAGGAGACAGCTGGCCATCGCTTGTAGCATTTGCGAGTGCCTGCGTGTGAGAC 150
 315 CCAAGAGACTGCAGCAGGAGGAGGACGATGGGCAAGTGGGCAACCTAGC 364
 151 CCAAGAGACTGCAGCAGGAGGAGGACGATGGGCAAGTGGGCAACCTAGC 200
 365 TTACTCTTGAACCCGGACCCCTCCACACAGACACGCTGCCCTGTACAT 414
 201 TTACTCTTGAACCCGGACCCCTCCACACAGACACGCTGCCCTGTACAT 250
 415 GAGAAAGGTGAAGAAATGCTAGAGGACTGATCCTGAGGCTCCTGAAAGT 464
 251 GAGAAGGTGAAGAAATGCTAGAGGACTGATCCTGAGGCTCCTGAAAGT 300
 465 CCAAGTACTTATGGGATGAGGAGCAGGACATGTTCAACCCCTGAACTCCA 514
 301 CCAAGTACTTATGGGATGAGGAGCAGGACATGTTCAACCCCTGAACTCCA 350
 515 GTTTAATGCCATCCAAAACGCTGGCAATCATTAACGTTCCCGCAGCGGCC 564
 351 GTTTAATGCCATCCAAAACGCTGGCAATCATTAACGTTCCCGCAGCGGCC 400
 565 TATTTTCCCAAGAGGACTCTAAACGCTCTAAACGCTG---TTCCGAAT 607
 401 TATTTTCCCAAGAGGACTCTAAAC--C---A---TGGATCCGAAT 436
 to: Aa111090 from 1 to 454
 CCM101_15537 x Aa111090 rev
 575 GAGGACTCTAAACGCTCTAAACGCTCTGTTGAAATGTCTCTCTGCCACAACTG 624
 1 GAGGACTCTAAACGCTCTAAACGCTCTGTTGAAATGTCTCTCTGCCACAACTG 50
 625 GGGCGGAGCTATGAAGCCTGGGTGGGAGAAAAGGACTTAGGAGAGGAAGT 674
 51 GGGCGGAGCTATGAAGCCTGGGTGGGAGAAAAGGACTTAGGAGAGGAAGT 100
 675 ATGGAGACAGAGGTTAGGGGTACAACTCTTCAGTGACAGTGGGTATCTTTT 724
 101 ATGGAGACAGAGGTTAGGGGTACAACTCTTCAGTGACAGTGGGTATCTTTT 150
 725 TGGTCCCTTAGGTGTGGTCTCCAGGCAATCTTTTCCCTTTGCCCCGGT 774
 151 TGGTCCCTTAGGTGTGGTCTCCAGGCAATCTTTTCCCTTTGCCCCGGT 200
 775 CGGCTTTGTACAAATGCCAAAAATCAAACTCTGCCCAAAGAGCGCG--AGT 822
 201 CGGCTTTGTACAAATGCCAAAAATCAAACTCTGCCCAAAGAGCGCGAGT 250
 823 TGGTGTAAAATGTATTTCTCTAGGGGCGACCTGCCCTCTCTCCCAAAC 872
 251 TGGTGTAAAATGTATTTCTCTAGGGGCGACCTGCCCTCTCTCCCAAAC 300
 873 CTCTGATGAAGACTCCAAGAGAAAGCTCAGACGAAGCTGTGATTCATG 922

15537





Waiting for Sarah Pollock

Sun Sep 21 17:13:21 1997

571 CCAAGAGGACTCTTAA 585
|||||
401 CCAAGAGGACTCTTAA 415
|||||

to: AA388536 from 1 to 554

CCM101_15537 x AA388536 rev ..

273 AGACAGCTGGCCATCGCTTACGATTTGACGTGCGTGTGAGACCCCAAGAGA 322
|||||

1 AGACAGCTGGCCATCGCTTACGATTTGACGTGCGTGTGAGACCCCAAGAGA 50
|||||

323 CTCACGAGGAGAGGAGGAGTGGGCAAGTGGGCAACCTAGCTTACCTTG 372
|||||

51 CTCACGAGGAGAGGAGGAGTGGGCAAGTGGGCAACCTAGCTTACCTTG 100
|||||

373 TAACCGGACCTCCACACACACACACACGCTGCGCTGTGACATGAGAAAG 422
|||||

101 TAACCGGACCTCCACACACACACACACGCTGCGCTGTGACATGAGAAAG 150
|||||

423 TGAAGGAATGCTAGAGGACTGATCTGAGGCTCTGAAAGTCCCAAGGTA 472
|||||

151 TGAAGGAATGCTAGAGGACTGATCTGAGGCTCTGAAAGTCCCAAGGTA 200
|||||

473 CTTATGGGATGAGGAGGAGGAGGAGGAGGAGGAGGAGGAGGAGGAGGAG 522
|||||

201 CTTATGGGATGAGGAGGAGGAGGAGGAGGAGGAGGAGGAGGAGGAGGAG 250
|||||

523 CCATCCAAAGCGTGGCAATCAATACCGTTCCCGGACGCGCTTATTTTC 572
|||||

251 CCATCCAAAGCGTGGCAATCAATACCGTTCCCGGACGCGCTTATTTTC 300
|||||

573 AAGAGGACTCTAAAGCTTAAAGCTGTTGGAATGCTCTGCCACAAC 622
|||||

301 AAGAGGACTCTAAAGCTTAAAGCTGTTGGAATGCTCTGCCACAAC 350
|||||

623 TGGGGCGGAGCTATGAAGCCTGGGTGGAGGAGGAGGAGGAGGAGGAGGAG 672
|||||

351 TGGGGCGGAGCTATGAAGCCTGGGTGGAGGAGGAGGAGGAGGAGGAGGAG 400
|||||

673 GTATGGAGACAGAGGTTAGGGGTACATATGCTTACAGTACAGTGGGTATCT 722
|||||

401 GTATGGAGACAGAGGTTAGGGGTACATATGCTTACAGTACAGTGGGTATCT 450
|||||

723 TTTGGTCCCTTAGGTGCTGCTTACGCGATTCCTTTTCCCTTTGCGCG 772
|||||

451 TTTGGTCCCTTAGGTGCTGCTTACGCGATTCCTTTTCCCTTTGCGCG 500
|||||

773 TGGGCTTGTACAAATGCCAAATAATCAAACTCTGCGCCAAAG-A-G--C 817
|||||

501 TGGGCTTGTACAAATGCCAAATAATCAAACTCTGCGCCAAAGTGGATC 549
|||||

818 CGAGT 822
|||

550 CGAAT 554
|||

to: AA270163 from 1 to 369

15537

Waiting for Sarah Pollock

Sun Sep 21 17:13:21 1997

CCM101_15537 x AA270163 rev ..

236 CACTCCATCTTTTGAATAGCAGATGCCACGGAGGAGAGCAGCTGGCCA 285
|||||

1 CACTCCATCTTTTGAATAGCAGATGCCACGGAGGAGAGCAGCTGGCCA 50
|||||

286 TCGCTTAGCATTTGCAAGTGCCTGAGACCCCAAGAGACTGCAGCAGGAG 335
|||||

51 TCGCTTAGCATTTGCAAGTGCCTGAGACCCCAAGAGACTGCAGCAGGAG 100
|||||

336 AGGACGAGTGGGCAAGTGGGCAACCTAGCTTACCTTGTAAACCGGACCT 385
|||||

101 AGGACGAGTGGGCAAGTGGGCAACCTAGCTTACCTTGTAAACCGGACCT 150
|||||

386 CCACACACACACACGCTGCGCTGTACATGAGAAAGTGAAGGAATGCT 435
|||||

151 CCACACACACACGCTGCGCTGTACATGAGAAAGTGAAGGAATGCT 200
|||||

436 AGAGGACTGATCTGAGGCTCTGAAAGTCCCAAGTACTTATGGGATGAG 485
|||||

201 AGAGGACTGATCTGAGGCTCTGAAAGTCCCAAGTACTTATGGGATGAG 250
|||||

486 GAGCAGGACATGTTCAACCTGAACTCCAGTAAATGCCATCCAAACGC 535
|||||

251 GAGCAGGACATGTTCAACCTGAACTCCAGTAAATGCCATCCAAACGC 300
|||||

536 TGGCAATCAATAACCGTTCGCGACGCGCTTATTTTCAAGAGGACTCTAA 585
|||||

301 TGGCAATCAATAACCGTTCGCGACGCGCTTATTTTCAAGAGGACTCTAA 350
|||||

586 ACCTCTAAACGCTCTGTTG 604
|||||

351 ACCTCTAAACGCTCTGTTG 369
|||||

to: AA274705 from 1 to 498

CCM101_15537 x AA274705 rev ..

73 GCTGACAAACACAGAGTACAGATTGCTCCCAACAGGGAGGAGCGCTTTCA 122
|||||

1 GCTGACAAACACAGAGTACAGATTGCTCCCAACAGGGAGGAGCGCTTTCA 50
|||||

123 TTGGTAGAGATCAATTGAAATGTCATGTGAGAAATGCAATGCTGCGCTC 172
|||||

51 TTGGTAGAGATCAATTGAAATGTCATGTGAGAAATGCAATGCTGCGCTC 100
|||||

173 CCCCCACAGAGTCCCTAAATGAAATGAAATGAAAGCCCCCATGAACT 222
|||||

101 CCCCCACAGAGTCCCTAAATGAAATGAAATGAAAGCCCCCATGAACT 150
|||||

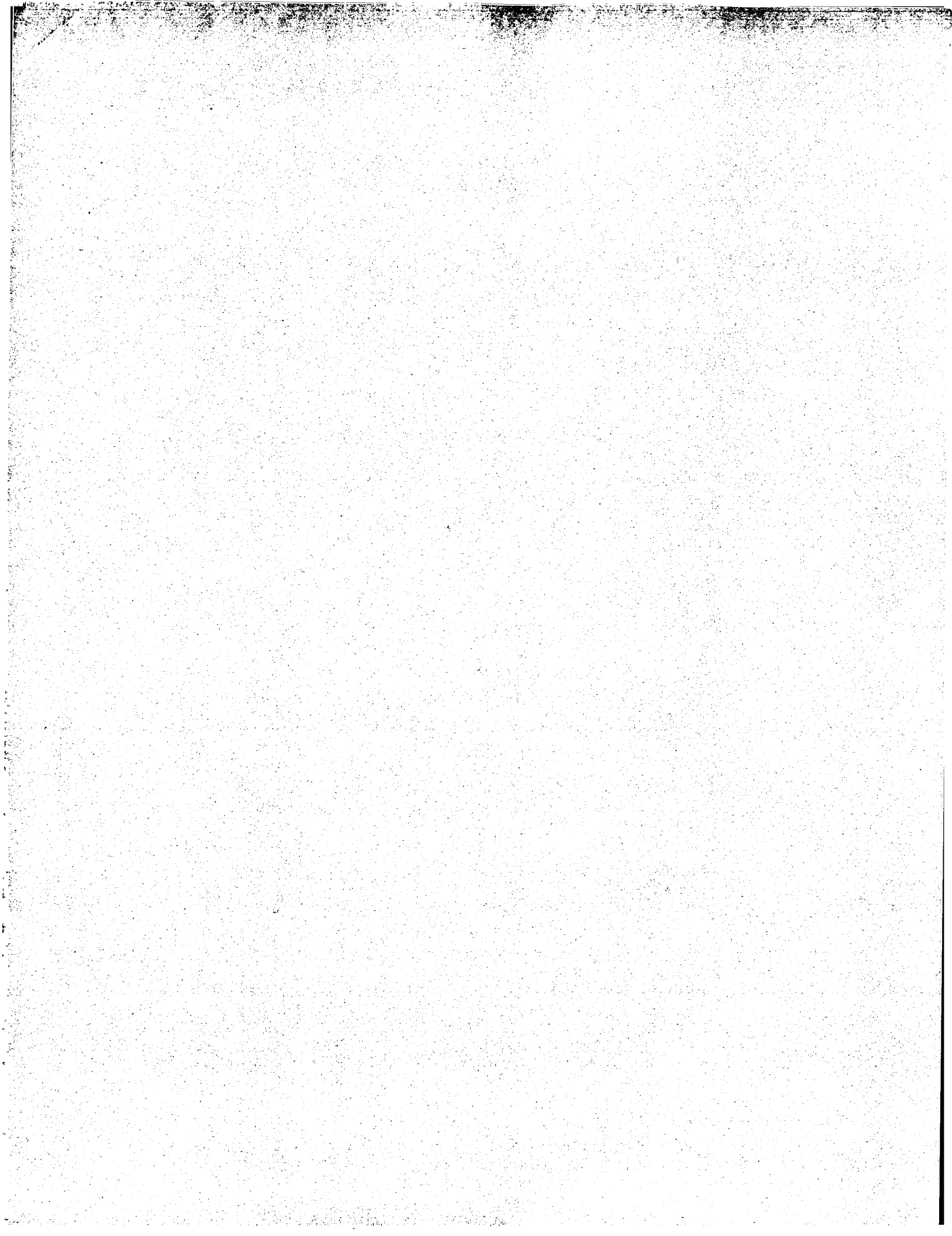
223 CTGGGCGCCCTCTCACTCCATCTTCTTTTGAATAGCAGATGCCACGAGGG 272
|||||

151 CTGGGCGCCCTCTCACTCCATCTTCTTTTGAATAGCAGATGCCACGAGGG 200
|||||

273 AGACAGCTGGCCATCGCTTAGCATTTTGCAGTGTGAGAGCCCAAGAGA 322
|||||

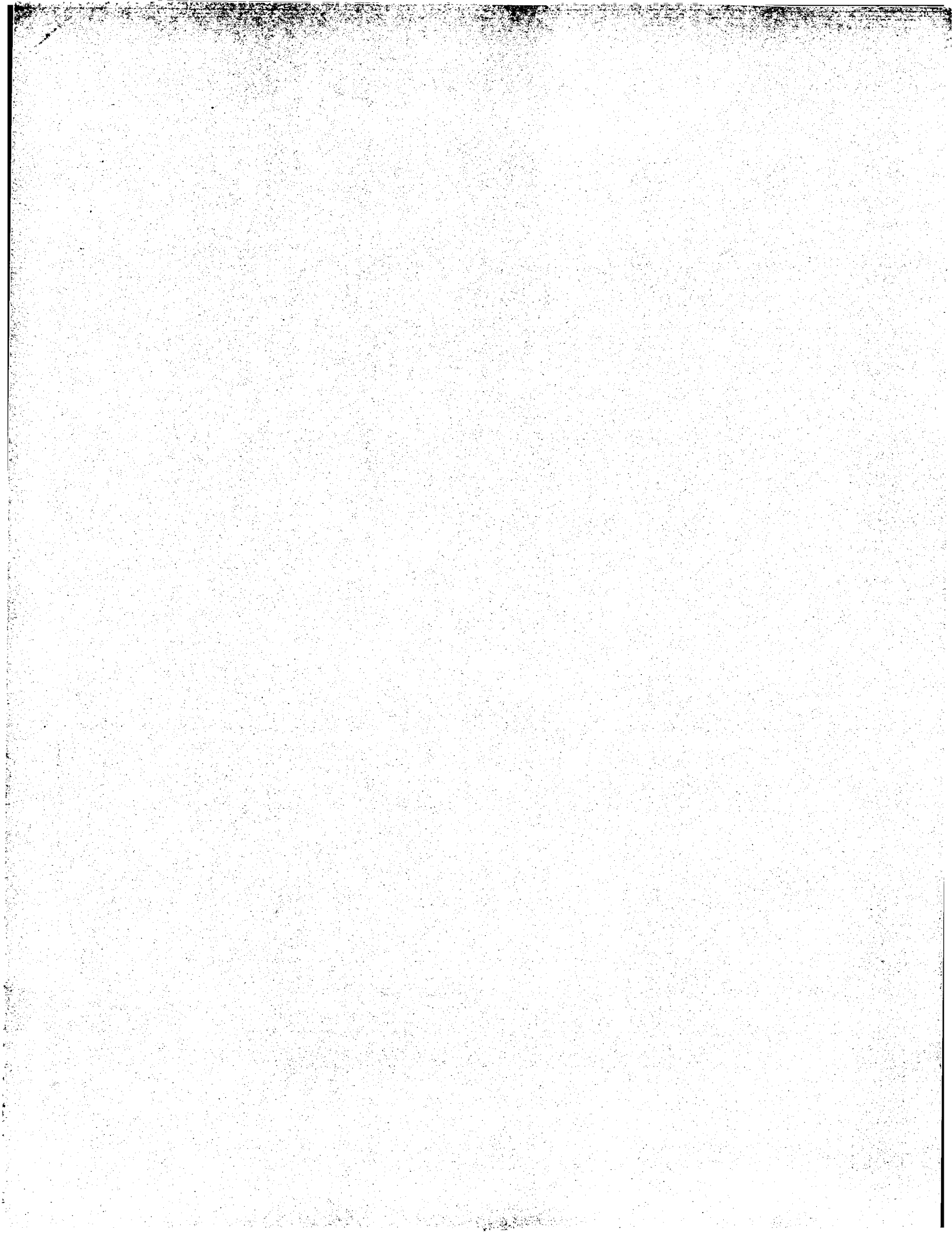
201 AGACAGCTGGCCATCGCTTAGCATTTTGCAGTGTGAGAGCCCAAGAGA 250
|||||

15537



CC 323 CTGCAGCAGGAGGAGGACGAGTGGGCAAGTGGGCAACCTAGCTTACCTTGG 372
CC
CC 251 CTGCAGCAGGAGGAGGAGGAGTGGGCAAGTGGGCAACCTAGCTTACCTTGG 300
CC
CC 373 TAACCCGGGACCTCCACACACAGACACGCTGCCCCCTGTACATGAGAAAGG 422
CC
CC 301 TAACCCGGGACCTCCACACACAGACACGCTGCCCCCTGTACATGAGAAAGG 350
CC
CC 423 TGAAGGAAATGCTAGAGGACTGATCTCTGAGGCTCTCTGAAAGTCCAAAGGTA 472
CC
CC 351 TGAAGGAAATGCTAGAGGACTGATCTCTGAGGCTCTCTGAAAGTCCAAAGGTA 400
CC
CC 473 CTATATGGGATGAGGAGCAGGACATGTTCAACCTGAACTCCAGTTTAATG 522
CC
CC 401 CTATATGGGATGAGGAGCAGGACATGTTCAACCTGAACTCCAGTTTAATG 450
CC
CC 523 CCATCCAAACCGTGGCAATCATACCGTTCCCG--CACGG--CCCTAT 567
CC
CC 451 CCATCCAAACCGTGGCAATCATACCGTTCCCGCATGGATCCGAAT 498
CC
CC to: AA212757 from 1 to 395
CC
CC CCM101_15537 x AA212757 rev ..
CC
CC 284 CATCGCTTAGCATTTGCAATGCGTGTGAGACCCCAAGAGACTGAGCAGGG 333
CC
CC 1 CATCGCTTAGCATTTGCAATGCGTGTGAGACCCCAAGAGACTGAGCAGGG 50
CC
CC 334 AGAGGACGAGTGGGCAAGTGGGCAACCTAGCTTACCTTGAACCCGGACC 363
CC
CC 51 AGAGGACGAGTGGGCAAGTGGGCAACCTAGCTTACCTTGAACCCGGACC 100
CC
CC 384 CTCCACACACAGACACGCTCCCTGTACATGAGAAAGTGAAGGAAATG 433
CC
CC 101 CTCCACACACAGACACGCTCCCTGTACATGAGAAAGTGAAGGAAATG 150
CC
CC 434 CTAGAGGACTGATCTGAGGCTCTCTGAAAGTCCAAAGGTAATGAGGATG 483
CC
CC 151 CTAGAGGACTGATCTGAGGCTCTCTGAAAGTCCAAAGGTAATGAGGATG 200
CC
CC 484 AGGACGAGACATGTTCAACCCCTGAACTCCAGTTTAAATGCCATCCAAAC 533
CC
CC 201 AGGACGAGACATGTTCAACCCCTGAACTCCAGTTTAAATGCCATCCAAAC 250
CC
CC 534 GCTGGCAATCATACCGTTCCCGCAGGCTCTATTTTCCAAAGGAGACTCT 583
CC
CC 251 GCTGGCAATCATACCGTTCCCGCAGGCTCTATTTTCCAAAGGAGACTCT 300
CC
CC 584 AAACGCTTAAACGCTGTGTCGAATGTCCTGCGCACAACTGGGGCGGAGC 633
CC
CC 301 AAACGCTTAAACGCTGTGTCGAATGTCCTGCGCACAACTGGGGCGGAGC 350
CC
CC 634 TATGAGGCTGGTGGGAAAGGAGCTAGGAGGAGGAGTATGG 678
CC
CC 351 TATGAGGCTGGTGGGAAAGGAGCTAGGAGGAGGAGTATGG 395
CC
CC to: AA388578 from 1 to 533
CC
CC CCM101_15537 x AA388578 rev ..
CC

CC 293 GCATTTTCAGTGCCTGTGAGACCCCAAGAGACTGACAGGAGGAGCGA 342
CC
CC 1 GCATTTTCAGTGCCTGTGAGACCCCAAGAGACTGACAGGAGGAGGAGCGA 50
CC
CC 343 GTGGGCAAGTGGGCAACCTAGCTTACCTTGTAAACCCGAGCCCTCCACACA 392
CC
CC 51 GTGGGCAAGTGGGCAACCTAGCTTACCTTGTAAACCCGAGCCCTCCACACA 100
CC
CC 393 CAGACACGCTGCCCTGTACATGAGAAAGGTTGAAGGAAATGCTAGAGGAC 442
CC
CC 101 CAGACACGCTGCCCTGTACATGAGAAAGGTTGAAGGAAATGCTAGAGGAC 150
CC
CC 443 TGATCTCTGAGGCTCTCTGAAAGTCCAAAGTCTTATGGGATGAGGAGCAGG 492
CC
CC 151 TGATCTCTGAGGCTCTCTGAAAGTCCAAAGTCTTATGGGATGAGGAGCAGG 200
CC
CC 493 ACATGTTTCAACCCCTGAACTCCAGTTTAAATGCCATCCAAACCGCTGGCAAT 542
CC
CC 201 ACATGTTTCAACCCCTGAACTCCAGTTTAAATGCCATCCAAACCGCTGGCAAT 250
CC
CC 543 CATACCCGTTCCCGCAGGCTCTATTTTCCAAAGGAGACTCTAAACGCTTA 592
CC
CC 251 CATACCCGTTCCCGCAGGCTCTATTTTCCAAAGGAGACTCTAAACGCTTA 300
CC
CC 593 AACGCTCTGTTGCAATGCTCTCTGCCACAACTGGGGCGGAGCTATGAAGCC 642
CC
CC 301 AACGCTCTGTTGCAATGCTCTCTGCCACAACTGGGGCGGAGCTATGAAGCC 350
CC
CC 643 TGGGTTGAAAGAAAGGACTAGGAGGAGGAGGATGAGGAGACAGAGGTTAGG 692
CC
CC 351 TGGGTTGAAAGAAAGGACTAGGAGGAGGAGGATGAGGAGACAGAGGTTAGG 400
CC
CC 693 GGTACAAATGCTTCAGTGACAGTGGGTATCTTTTGGTCCCTTAGGTGTGT 742
CC
CC 401 GGTACAAATGCTTCAGTGACAGTGGGTATCTTTTGGTCCCTTAGGTGTGT 450
CC
CC 743 CCGTACGGCAATCTTTTCCCTTTGCGGCTGCGGCTTTGTACAAATGCCA 792
CC
CC 451 CCGTACGGCAATCTTTTCCCTTTGCGGCTGCGGCTTTGTACAAATGCCA 499
CC
CC 793 ABAATCAAACCTCTGCCCAAGG-A-G---CCGA- 820
CC
CC 500 CCGTCAAACCTCTGCCCAAGGAGGATGCCGA 533
CC
CC to: AA023109 from 1 to 466
CC
CC CCM101_15537 x AA023109 rev ..
CC
CC 142 ATGTCTATGTGAGAAATGCAATGCTGCGCTCCCGCCACAGAGTCCCTAAA 191
CC
CC 1 ATGTCTATGTGAGAAATGCAATGCTGCGCTCCCGCCACAGAGTCCCTAAA 50
CC
CC 192 TGAATGAAATGATGATAAGCCCGCCCATGAACTCTGGGCGCCCTCTCCTCC 241
CC
CC 51 TGAATGAAATGATGATAAGCCCGCCCATGAACTCTGGGCGCCCTCTCCTCC 100
CC
CC 242 ATCTTCTTTGTAATAGCAGATGCCACGAGGAGGAGACAGCTGGCCATCGCTT 291
CC



Liliana P. Sarah Pollock

Submitted 11/17/13 2:11 PM

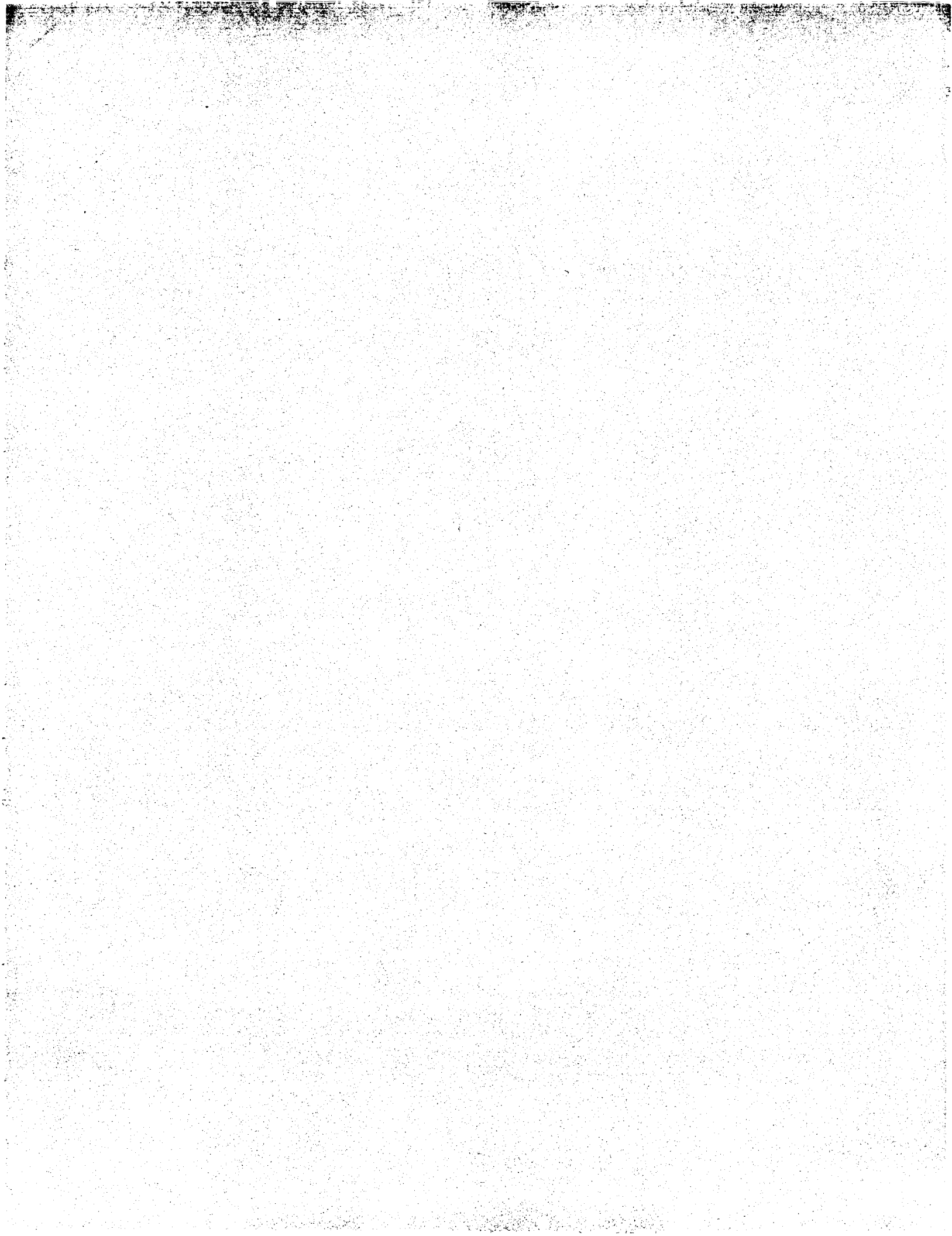
101 ATCTTCTTTTGAATAGCAGATGCCACGGAGGAGACAGCTGCCCATCGCTT 150
 292 ACATTTCCAGTGGGTGTGAGACCCAGAGACTCCAGCAGGAGAGAGCG 341
 151 AGCATTTTCAGTGGGTGTGAGACCCAGAGACTCCAGCAGGAGAGAGCG 200
 342 AGTGGGCAAGTGGGCAACCTAGCTTACCTTGTAAACCCGGACCCCTCCACAC 391
 201 AGTGGGCAAGTGGGCAACCTAGCTTACCTTGTAAACCCGGACCCCTCCACAC 250
 392 ACAGACACGCTGCCCTGTACATGAGAAAGGTGAAGGAAATGCTAGAGGA 441
 251 ACAGACACGCTGCCCTGTACATGAGAAAGGTGAAGGAAATGCTAGAGGA 300
 442 CTGATCTCTGAGGCTCTGAAAGTCCAAAGTACTTATGGGATGAGGAGCAG 491
 301 CTGATCTCTGAGGCTCTGAAAGTCCAAAGTACTTATGGGATGAGGAGCAG 350
 492 GACATGTTCAACCTGACTCCAGTTTAAATGCCATCCAAACCGCTGGCAA 541
 351 GACATGTTCAACCTGACTCCAGTTTAAATGCCATCCAAACCGCTGGCAA 400
 542 TCATTAACGTTCCCGACCGCCCTATTTTCCAAAGGAGCTCTAAACGCTCT 591
 401 TCATTAACGTTCCCGACCGCCCTATTTTCCAAAGGAGCTCTAAACGCTCT 450
 592 AAACGCTCTGTTCCGAAT 607
 451 AAACGCTCTGTTCCGAAT 466

PPS end.

Sequence 1116 BP: 298 A; 293 C; 272 G; 253 T; 0 other;
 agacttttaa ggaagcaat tattttattt cctggcctg acagcattga cttcaaaagta 60
 ccagaacctg agctgacaa acagacgta cagattgctc ccaacagggga ggaactcttt 120
 catggtaga gatgaattga aatgctatgt gagaatgcaa tgcctggccc tccccccaca 180
 gactccctaa atgaaatgaa tgaataaagc cccatgaaa ctctggcgcc cctctcactc 240
 catctctttt gaatagcaga tgcacggag ggaagagact ggcctatgct tagcatttgc 300
 agtgcgtgtg agaccacaaga gactgcaga ggaagagagac gtggggcaca gtgggcaacc 360
 tagcttacct tataaccgg accctcaca cacagacacg ctgcccctgt acatgagaaa 420
 ggtgaagaaa atgctagagg actgactctg aggtcctga aagtcacaag tacttatggg 480
 atgaggagca ggaatgttc aacctgaac tccagtttaa tgcacacaa aacgtgga 540
 atcataaccg tcccgcacg gccctatttt ccaagagagac tctaaacgtc taacgtctg 600
 ttcgaatgct tctgcacaca actggggcgg agctatgaag cctgggtgga agaaaaggac 660
 tagagagga aggtatggag acagaggtta ggggtacaat gcttcagta cagtgggtat 720
 ctttggctc cttaggtgtg gtccctacgg catcttttc ccccttggc cgtcggtgt 780
 gtacaaatgc caaaatacaa acctctggcc aagagagcga gttgctgta aattgtatc 840
 tctcttagcg gccacctgcc ctctctccaa acctctgatg aaactgcacg agagaaagct 900
 caccgaag cttgtagtca tgcgtccccc atcccgtct caccgcaat gaaacacaa 960
 tctctcgg gctgaggaag cagagaaaag taacaattga gcttgtaaa aaagacgac 1020
 agcgtgacg gttgtagtt tcccagatgc ttcacottgc gtctctcca tgggtggag 1080
 cctctccctct agttccccaa gtccatggaat ccgaat 1116

//

1537



Shirley, Sarah Pollock

Sim Sep 21 17:13:26 1997

```
>CCM101_15537.0 Mus musculus (mouse) CCM101_15537
agacttttaaggtaagcaattattttatttccctggcctgacagcatigacttcaagta
ccagaacctgaagtgacaaaacagacgtacagatgctcccaacaggaggagcgccttt
cattgtagagatgaattgaatgcatgtagaagtgcaatgctgccccccccccaca
gagtccttaaatgaatgaatgaatgaatgaatgaatgaatgaatgctgccccctcactc
catctcttgaatagcagatgccacggaggagacagctggccatcgcttagcatitgc
agtgcgtgtagaccacaagactgcagcaggaggagcagtgaggcaagtgggcaacc
tagcttacctgttaaccggaccctccacacagacacgctgccccctgtacatgagaaa
ggtgaagaaaatgctagaggactgactgagggctcctgaaagtccaaggtactta tggg
atgaggagcaggacatgttcaacctgaactccagtttaattgctcccaaacacgtggca
atcataaccttccgcacgcccctatttcccaaggagactctaaactcaaacgtctg
ttcgaatgtctctgccaaactggggcggagctatgaagcctgggtggaagaaaaaggac
taggagagggaaggtatggagacagagggttaggggtacaaatgcttcagtgacag tgggtat
cttttggctcccttaggtggtgctcctacgcatcttttccccctttgccccgtgggtt
gtacaatgccaaaatcaaacctctgccaaaaggccaggttgctgctaaattgtatc
ttcttagcggccacctgccccctcccaactctgtagaaactgacacgacgagagaaagt
cacagaagctgtgattcatgtgctgctccatccctctcacagccaatgcaacacaag
ttcttctgggctgagggaagcagagaaaagtaacaattgagcttggaaaaaaagacgac
agcgtgctgggtggtgttcccgatgcttccacttgcgtctctccatgggttgggag
ccctccctctggttcccaagtccatggatccgaat
```

15537.transcript

Listing for Sarah Pollock Sun Sep 21 17:13:34 1997

ID 19101 cluster; DNA; EST; 701 BP.
AC CCM101_19101
DT 24-AUG-1997 (Rel. 101, created)
DE Mus musculus (mouse)
OS functional information, bla, bla, bla 4 6 0
CC Alignment of CCM101_19101 to: it's cluster
CC
CC
CC

list of EST and RNA names, tissues (uniq -i -c) etc.
(any number of lines with html text).

Sequence	Strd	Partial Length	Tissue	Documentation
CCM101_19101_0	+	no	430	transcript ++
CCM101_19101_1	+	no	312	transcript ++
CCM101_19101_2	+	no	465	transcript --
CCM101_19101_3	+	no	347	transcript --
W47953	+	no	430	EST
W50479	+	no	315	EST
W48543	+	no	427	EST
AA389214	+	yes	326	EST
AA451285	+	no	420	EST
W48549	+	no	430	EST

to: CCM101_19101_0 from 1 to 430

CCM101_19101 x CCM101_19101_0 ..

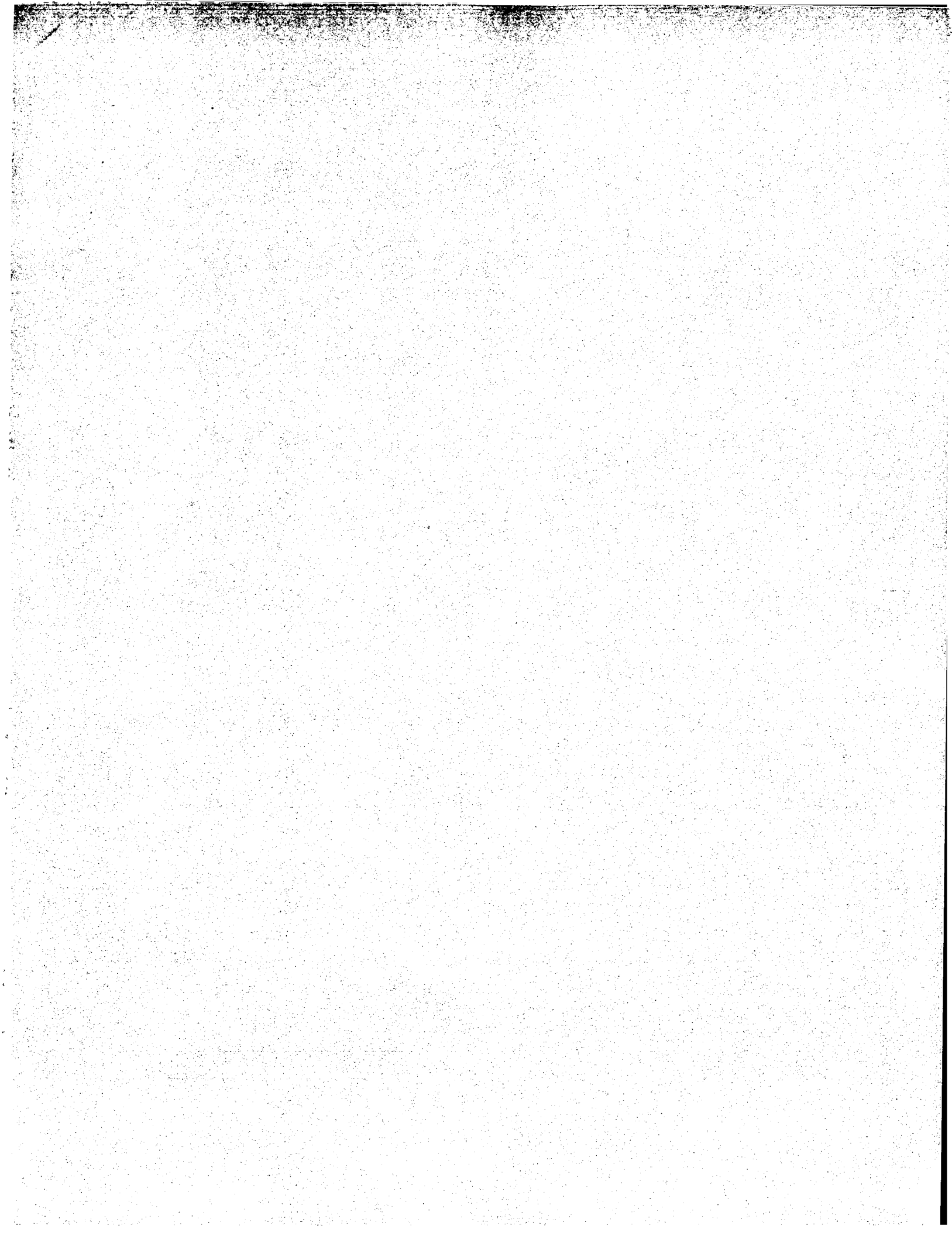
```
1 TCACTGCTCTTTAGTCTCTTTGGACACTTTCCAAATATCGGGCGATGA 50
+++++
1 TCACTGCTCTTTAGTCTCTTTGGACACTTTCCAAATATCGGGCGATGA 50
+++++
51 CCGTCAAAACCGAGGCTGCTCGAAGTACCTCACCCTACTCCAGAATGAGG 100
+++++
51 CCGTCAAAACCGAGGCTGCTCGAAGTACCTCACCCTACTCCAGAATGAGG 100
+++++
101 GGAATGGTACCAATCTCATATGTCGCGCTGCGCGCTGCGCGCTCGG 150
+++++
101 GGAATGGTACCAATCTCATC----- 121
151 TCCGCTATATGGGCGAGATGACAGGCGGCTGCGGCTCGGCTCGA 200
122 ----- 121
201 GTCCCTGCTCGGCGCGCACAAAAGCGGCGGAGGCTCAGAAAAGGA 250
122 ----- 121
251 GCGAGTCGCTCTGCTGAGCGGACTTTCATGAAACAGAGAAGGAT 300
+++++
122 -----GCTTTCATGAAACAGAGAAGGAT 144
301 GGGGCTGAGCAATTTATTCAGAGCTTGCCCAACAATCTCATGATGCA 350
+++++
145 GGGGCTGAGCAATTTATTCAGAGCTTGCCCAACAATCTCATGATGCA 194
```

19101

Listing for Sarah Pollock Sun Sep 21 17:13:34 1997

```
351 AACGTAAAGTTTGGGTTATTTGGGAAAAATTTGAAAGAAATCGTTTAAATTTA 400
+++
195 AAC----- 197
401 CTAGCAGCTAAACTTGTTCACGTTGGCGAGGGGTGGGGAGCTGAAGT 450
198 ----- 197
451 CTTTCTCTCCATGTGTAGTACCTCTGAAGTTCAATCTCTATTTGAAATCTCC 500
+++++
198 -----ACCTGGAAGTTCAATCTCTATTTGAAATCTCC 229
501 CAACCTCAGAGCCCGAAGCTTATGAACGCCAACCCCTCACCTCTCTCCAAG 550
+++++
230 CAACCTCAGAGCCCGAAGCTTATGAACGCCAACCCCTCACCTCTCTCCAAG 279
551 TCCCTCTCAAAATAATCAACCTGGGTCCATCTCAATCCCAATCCCAAGCCAAAC 600
+++++
280 TCCCTCTCAAAATAATCAACCTGGGTCCATCTCAATCCCAAGCCCAAG 329
601 CTTCTGACTTCCACTTCTTTGAAAGTGTGCGAAAGGCGAGTTTGGAAAG 650
+++++
330 CCTCTGACTTCCACTTCTTTGAAAGTGTGCGAAAGGCGAGTTTGGAAAG 379
651 GTTCTTCTAGCAAGGACAAAGGAGAGAGCAATCTATGCGCTCAAGT 700
+++++
380 GTTCTTCTAGCAAGGACAAAGGAGAGAGCAATCTATGCGCTCAAGT 429
701 T 701
430 T 430
to: CCM101_19101_1 from 1 to 312
CCM101_19101 x CCM101_19101_1 ..
1 TCACTGCTCTTTAGTCTCTTTGGACACTTTCCAAATATCGGGCGATGA 50
+++++
1 TCACTGCTCTTTAGTCTCTTTGGACACTTTCCAAATATCGGGCGATGA 50
+++++
51 CCGTCAAAACCGAGGCTGCTCGAAGTACCTCACCCTACTCCAGAATGAGG 100
+++++
51 CCGTCAAAACCGAGGCTGCTCGAAGTACCTCACCCTACTCCAGAATGAGG 100
+++++
101 GGAATGGTACCAATCTCATGTCGCGCTGCGCGCTGTCGCGCTCGG 150
+++++
101 GGAATGGTACCAATCTCATC----- 121
151 TCCGCTATATGGGCGAGATGACAGGCGGCTGCGGCTCGGCTCGA 200
122 ----- 121
201 GTCCCTGCTCGGCGCGCACAAAAGCGGCGGAGGCTCAGAAAAGGA 250
122 ----- 121
251 GCGAGTCTCTGCTGAGCGGACTTTCATGAAACAGAGAAGGAT 300
+++++
+++++
```

19101



Sun Sep 21/17/1997

Listing for Sarah Pollock

```

CC CC
122 -----GCTTTCATGAACAGAGAGGAT 144
CC CC
301 GGGCCTGAACGATTTTATTCAGAGCTTGCCAAACACTCTCTATGATGCA 350
CC CC
145 GGGCCTGAACGATTTTATTCAGAGCTTGCCAAACACTCTCTATGATGCA 194
CC CC
351 AACGTAACTTTTGGGTTATTTGGGAAATTCGAAAGAACTCTTTAAATTTA 400
CC CC
195 AACGTAACTTTTGGGTTATTTGGGAAATTCGAAAGAACTCTTTAAATTTA 244
CC CC
401 CTAGCAGCTAAACTTTTTCACGCTGGCGAGGGGGTGGGGAGCTGAAGT 450
CC CC
245 CTAGCAGCTAAACTTTTTCACGCTGGCGAGGGGGTGGGGAGCTGAAGT 294
CC CC
451 CTTTCTCCATGATAGAT 468
CC CC
295 CTTTCTCCATGATAGAT 312
CC CC
to: CCM101_19101_2 from 1 to 465
CCM101_19101 x CCM101_19101_2 ..
CC CC
122 GTCGCCCTGCCCGCTGTCGCCCTCGGTCGGATCTATGGCGAGATG 171
CC CC
1 GTCGCCCTGCCCGCTGTCGCCCTCGGTCGGATCTATGGCGAGATG 50
CC CC
172 CAGGGCGGCTGCGGCTCGGCTCGAGTCCCTGCTCGGCGCGCA 221
CC CC
51 CAGGGCGGCTGCGGCTCGGCTCGAGTCCCTGCTCGGCGCGCA 100
CC CC
222 CAAAAAGCGGGGAGGCTCAGAAAGGAGGAGTCCGCTCGTGAAGG 271
CC CC
101 CAAAAAGCGGGGAGGCTCAGAAAGGAGGAGTCCGCTCGTGAAGG 150
CC CC
272 GACTGGGCTTTCATGAACAGAGAGGAGTGGGCTGAACGATTTTATTTCA 321
CC CC
151 GACTGGGCTTTCATGAACAGAGAGGAGTGGGCTGAACGATTTTATTTCA 200
CC CC
222 CAAAAAGCGGGGAGGCTCAGAAAGGAGGAGTCCGCTCGTGAAGG 271
CC CC
101 CAAAAAGCGGGGAGGCTCAGAAAGGAGGAGTCCGCTCGTGAAGG 150
CC CC
272 GACTGGGCTTTCATGAACAGAGAGGAGTGGGCTGAACGATTTTATTTCA 321
CC CC
151 GACTGGGCTTTCATGAACAGAGAGGAGTGGGCTGAACGATTTTATTTCA 200
CC CC
322 GAAGCTTGCACAACTCTATGATGCAAAACGTAAGTTTGGGTTATG 371
CC CC
201 GAAGCTTGCACAACTCTATGATGCAAAACGTAAGTTTGGGTTATG 232
CC CC
372 GGGAAATTTGAAAGAACTGTTTAAATTTACTAGCAGCTAAACCTGTTTCA 421
CC CC
233 -----
CC CC
422 CGGTGGCGAGGGGTGGGGAGCTGAAGTCTTTCCTCCATGATAGAT 471
CC CC
233 -----ACC 235
CC CC
472 CTGAAGTTCACTCTATTTGAAATCTCCAACTCAGAGCGCCGAACTT 521
CC CC
236 CTGAAGTTCACTCTATTTGAAATCTCCAACTCAGAGCGCCGAACTT 285
CC CC
522 ATGAAGCGCAACCTCTCAAGTCCCTGCTCTCAACAAATCAACCT 571
CC CC
286 ATGAAGCGCAACCTCTCAAGTCCCTGCTCTCAACAAATCAACCT 335
CC CC
572 GGTGTCATCTCTAAATCCCAACGCAACCTCTGACTTCCACTTTTGA 621

```

19101

Sun Sep 21/17/1997

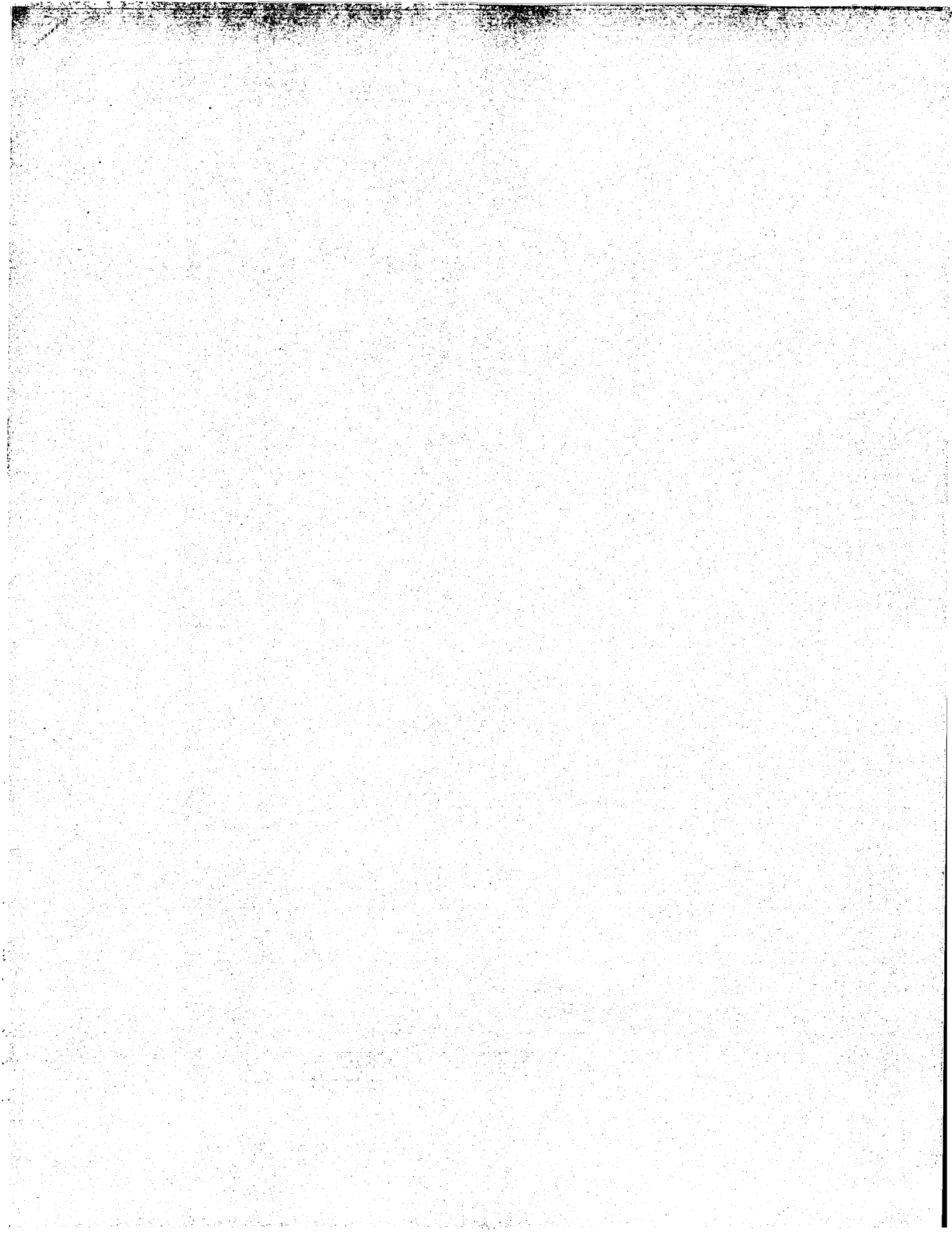
Listing for Sarah Pollock

```

CC CC
336 GGTTCATCTCTCAATCCCGACGCAACCCCTCTGACTTCCACTTCTTGA 385
CC CC
622 AAGTGATCGGAAAGGAGGTTTGGAAAGGTTCTTCTAGCAAGGACAAAG 671
CC CC
386 AAGTGATCGGAAAGGAGGTTTGGAAAGGTTCTTCTAGCAAGGACAAAG 435
CC CC
672 GCAGAAAGACATCTATGCGCTCAAAAGTT 701
CC CC
436 GCAGAAAGACATCTATGCGCTCAAAAGTT 465
CC CC
to: CCM101_19101_3 from 1 to 347
CCM101_19101 x CCM101_19101_3 ..
CC CC
122 GTCGCCCTGCCCGCTGTCGCCCTCGGTCGGATCTATGGCGAGATG 171
CC CC
1 GTCGCCCTGCCCGCTGTCGCCCTCGGTCGGATCTATGGCGAGATG 50
CC CC
172 CAGGGCGGCTGCGGCTCGGCTCGAGTCCCTGCTCGGCGCGCA 221
CC CC
51 CAGGGCGGCTGCGGCTCGGCTCGAGTCCCTGCTCGGCGCGCA 100
CC CC
222 CAAAAAGCGGGGAGGCTCAGAAAGGAGGAGTCCGCTCGTGAAGG 271
CC CC
101 CAAAAAGCGGGGAGGCTCAGAAAGGAGGAGTCCGCTCGTGAAGG 150
CC CC
272 GACTGGGCTTTCATGAACAGAGAGGAGTGGGCTGAACGATTTTATTTCA 321
CC CC
151 GACTGGGCTTTCATGAACAGAGAGGAGTGGGCTGAACGATTTTATTTCA 200
CC CC
322 GAAGCTTGCACAACTCTATGATGCAAAACGTAAGTTTGGGTTATG 371
CC CC
201 GAAGCTTGCACAACTCTATGATGCAAAACGTAAGTTTGGGTTATG 250
CC CC
372 GGGAAATTTGAAAGAACTGTTTAAATTTACTAGCAGCTAAACCTGTTTCA 421
CC CC
251 GGGAAATTTGAAAGAACTGTTTAAATTTACTAGCAGCTAAACCTGTTTCA 300
CC CC
422 CGGTGGCGAGGGGTGGGGAGCTGAAGTCTTTCCTCCATGATAGAT 468
CC CC
301 CGGTGGCGAGGGGTGGGGAGCTGAAGTCTTTCCTCCATGATAGAT 347
CC CC
to: W47953 from 1 to 430
CCM101_19101 x W47953 ..
CC CC
1 TCACCTGCTCTCTTAGTCTCTTTGGACACATTTCCAAATATCGGGCGATGA 50
CC CC
1 TCACCTGCTCTCTTAGTCTCTTTGGACACATTTCCAAATATCGGGCGATGA 50
CC CC
51 CGGTCAAAACCGAGGGGTGCTCGAAGTACCCTCACTACTCCAGATGAGG 100
CC CC
51 CGGTCAAAACCGAGGGGTGCTCGAAGTACCCTCACTACTCCAGATGAGG 100
CC CC
101 GGAATGATGATCAATCTCTCATGCTGCGCGCTGCTGCGGCCCTCGG 150
CC CC
101 GGAATGATGATCAATCTCTCATGCTGCGCGCTGCTGCGGCCCTCGG 121

```

19101



Listing for Sarah Pollock

Sun Sep 2 17:33:41 1997

```

CC
CC
CC
151 TCCGATCTATGGCGAGATGACAGGCGCGTGGCTCGGCTCGGCTCGA 200
CC
CC
122 ----- 121
CC
CC
201 GTCCCTGCTCCGGCCCGCCCAAAAAGCGGGGAGGCTCAGAAAAGGA 250
CC
CC
122 ----- 121
CC
CC
251 GCGAGTCCGCTGCTGAGGAGCTGGGCTTTTCATGAACAGAGAGAT 300
CC
CC
122 ----- 144
CC
CC
301 GGGCTGAACGATTTTATTCAGAGCTTGCACAACTCTCTATGATGCA 350
CC
CC
145 GGGCTGAACGATTTTATTCAGAGCTTGCACAACTCTCTATGATGCA 394
CC
CC
351 AACGTAAGTTTGGGTTATTCGGGAAATTTGAAAGAAATCGTTTAAATTA 400
CC
CC
195 AAC----- 197
CC
CC
401 CTAGCAGCTAAACTTGTTCACGCTGGCGAGGGGTGGGGAGCTGAAGT 450
CC
CC
198 ----- 197
CC
CC
451 CTTCCTCCATGGTAGATACCTGAACTTCAATCTTATTTGAAATCTCC 500
CC
CC
198 ----- 229
CC
CC
501 CAACCTCAGAGCCGAACTTATGACGCGCAACCTCCTCCTCCCAAG 550
CC
CC
230 CAACCTCAGAGCCGAACTTATGACGCGCAACCTCCTCCTCCCAAG 279
CC
CC
551 TCCCTCTCAACAACTCAACCTGGCTCCATCTCAATCCCAACGCAAAAC 600
CC
CC
280 TCCCTCTCAACAACTCAACCTGGCTCCATCTCAATCCCAACGCAAAAC 329
CC
CC
601 CTCTGACTTCCACTTCTTGAAGTGTATGGAAGGAGGAGGATTTTGAAG 650
CC
CC
330 CTCTGACTTCCACTTCTTGAAGTGTATGGAAGGAGGAGGATTTTGAAG 379
CC
CC
651 GTTCTTCTAGAGGACAGGAGGAGGAGGAGGATTTCTATGCGCTCAAGT 700
CC
CC
380 GTTCTTCTAGAGGACAGGAGGAGGAGGAGGATTTCTATGCGCTCAAGT 429
CC
CC
701 T 701
CC
CC
430 T 430
CC
CC
to: W50479 from 3 to 315
CC
CC
CCM101_19101 x W50479
CC
CC
1 TCACTGCTCTTTAGTCTCTTTGGACACTTTCCAAATATCGGGCGATGA 50
CC
CC
3 TCACTGCTCTTTAGTCTCTTTGGACACTTTCCAAATATCGGGCGATGA 52
CC
CC
51 CCGTCAAAACCGAGGCTGCTCGAAGTACCTCACTACCTCGGCTCGGCTCGA 100
CC
CC
*****
CC
CC
51 CCGTCAAAACCGAGGCTGCTCGAAGTACCTCACTACCTCGGCTCGGCTCGA 100
CC
CC
*****
CC
CC
101 GGAATGGTAGCAATCTCTCATCGTTCGCGCTCGGCTCGGCTCGGCTCGG 150
CC
CC
*****
CC
CC
101 GGAATGGTAGCAATCTCTCATC----- 121
CC
CC
151 TCCGCATCTATGGCGAGATGACAGGCGCGCTGGCTCGGCTCGGCTCGA 200
CC
CC
122 ----- 121
CC
CC
201 GTCCCTGCTCCGGCCCGCCCAAAAAGCGGGGAGGCTCAGAAAAGGA 250
CC
CC
122 ----- 121
CC
CC
251 CCGTCAAAACCGAGGCTGCTCGAAGTACCTCACTACCTCGGCTCGGCTCGG 300
CC
CC
*****

```

19101

Listing for Sarah Pollock

Sun Sep 2 17:33:41 1997

```

CC
CC
CC
53 CCGTCAAAACCGAGGCTGCTCGAAGTACCTCACTACCTCGGCTCGGCTCGA 102
CC
CC
101 GGAATGGTAGCAATCTCTCATCGTTCGCGCTCGGCTCGGCTCGGCTCGG 150
CC
CC
103 GGAATGGTAGCAATCTCTCATC----- 123
CC
CC
151 TCCGCATCTATGGCGAGATGACAGGCGCGCTGGCTCGGCTCGGCTCGA 200
CC
CC
124 ----- 123
CC
CC
201 GTCCCTGCTCCGGCCCGCCCAAAAAGCGGGGAGGCTCAGAAAAGGA 250
CC
CC
124 ----- 123
CC
CC
251 GCGAGTCCGCTCTGCTGAGCGGACTGGGCTTTTCATGAACAGAGAGAT 300
CC
CC
124 ----- 146
CC
CC
301 GGGCTGAACGATTTTATTCAGAGCTTGCACAACTCTCTATGATGCA 350
CC
CC
147 GGGCTGAACGATTTTATTCAGAGCTTGCACAACTCTCTATGATGCA 196
CC
CC
351 AAC-CTAAGTTTGGGTTATTCAGAGCTTGCACAACTCTCTATGATGCA 399
CC
CC
197 AACAGTAAAGTTTGGGTTATTCAGAGCTTGCACAACTCTCTATGATGCA 246
CC
CC
400 ACTAGCAGCTAAACTTGTTCACGCTGGCGAGGGGTGGGGAGCTGAAG 449
CC
CC
247 ACTAGCAGCTAAACTTGTTCACGCTGGCGAGGGGTGGGGAGCTGAAG 296
CC
CC
450 TCTTCTCTCCATGCTAGAT 468
CC
CC
297 TCTTCTCTCCATGCTAGAT 315
CC
CC
to: W48543 from 1 to 427
CC
CC
CCM101_19101 x W48543
CC
CC
1 TCACTGCTCTCTTTAGTCTCTTTGGACACTTTCCAAATATCGGGCGATGA 50
CC
CC
1 TCACTGCTCTCTTTAGTCTCTTTGGACACTTTCCAAATATCGGGCGATGA 50
CC
CC
51 CCGTCAAAACCGAGGCTGCTCGAAGTACCTCACTACCTCGGCTCGGCTCGA 100
CC
CC
51 CCGTCAAAACCGAGGCTGCTCGAAGTACCTCACTACCTCGGCTCGGCTCGA 100
CC
CC
101 GGAATGGTAGCAATCTCTCATCGTTCGCGCTCGGCTCGGCTCGGCTCGG 150
CC
CC
*****
CC
CC
101 GGAATGGTAGCAATCTCTCATC----- 121
CC
CC
151 TCCGCATCTATGGCGAGATGACAGGCGCGCTGGCTCGGCTCGGCTCGA 200
CC
CC
122 ----- 121
CC
CC
201 GTCCCTGCTCCGGCCCGCCCAAAAAGCGGGGAGGCTCAGAAAAGGA 250
CC
CC
122 ----- 121
CC
CC
251 GCGAGTCCGCTCTGCTGAGCGGACTGGGCTTTTCATGAACAGAGAGAT 300
CC
CC
*****

```

19101

Listing for Sarah Pollock

Sun Sep 2 17:33:41 1997

19101

Listing for Sarah Pollock Sun Sep 21/17:13:34:1997

122 -----GCTTTTCATGAAACAGAGAAGAT 144
 CC
 CC
 CC
 301 GGGCCTGACGATTTTATTCAGAGCTTGCACAACTCTTATGATGCA 350
 CC
 CC
 CC
 145 GGGCCTGACGATTTTATTCAGAGCTTGCACAACTCTTATGATGCA 194
 CC
 CC
 CC
 351 AACGTAAGTTTGGGTATTCGGGAAATTTGAAAGATCGTTTAAATTTA 400
 CC
 CC
 CC
 195 AAC----- 197
 CC
 CC
 CC
 401 CTAGCAGCTAAACCTTGTTCACGTGGCAGGGGGTGGGGAGCTGAAGT 450
 CC
 CC
 CC
 198 ----- 197
 CC
 CC
 CC
 451 CTTTTCCTCATGTAGATACCTGAGTTCATTCCTATTTGAAATCTCC 500
 CC
 CC
 CC
 198 -----ACCTGAGTTCATCTCTTATTTGAAATCTCC 229
 CC
 CC
 CC
 501 CAACCTCAGAGCCGGAATTTATGAGCGCAACCTCTACCTCTCCCAAG 550
 CC
 CC
 CC
 230 CAACCTCAGAGCCGGAATTTATGAGCGCAACCTCTACCTCTCCCAAG 279
 CC
 CC
 CC
 551 TCCTCTCAACAAATCAACCTGGGTCTCAATTCCTCAATTCCTCCCAAG 600
 CC
 CC
 CC
 280 TCCTCTCAACAAATCAACCTGGGTCTCAATTCCTCAATTCCTCCCAAG 329
 CC
 CC
 CC
 601 CCTCTGACTTCCACTCTTGAAGTATCGGAAAGGAGTTTGGAAAG 650
 CC
 CC
 CC
 330 CCTCTGACTTCCACTCTTGAAGTATCGGAAAGGAGTTTGGAAAG 379
 CC
 CC
 CC
 651 GTTCTTCTAGCAAGCACAAGCAGAGAGCAATTTCTATGCCGTCAA 698
 CC
 CC
 CC
 380 GTTCTTCTAGCAAGCACAAGCAGAGAGCAATTTCTATGCCGTCAA 427
 CC
 CC
 CC
 to: AA389214 from 7 to 326
 CCM101_19101 x AA389214
 1 TCACTGCT-CCTTTAGTCTC-TTGGACACTTTCCAAATATCGGGCGAT 48
 CC
 CC
 CC
 7 TCACTGCTCCTCAGTCTCTTTTGGGCTCTTTCCGGGCAATCGGGAGAT 56
 CC
 CC
 CC
 49 GACCGTCAAAACCGAGGCTCTGAAAGTACCTCTACCTCTCCAGATGA 98
 CC
 CC
 CC
 57 GACCGTCAAAACCGAGGCTCTGAAAGTACCTCTACCTCTCCAGATGA 106
 CC
 CC
 CC
 99 GGGGAATGTAGCAATCTCTATGCTCGCGTGGCGGCTGTGGCGCCCTC 148
 CC
 CC
 CC
 107 GGGGAATGTAGCAATCTCTATC----- 129
 CC
 CC
 CC
 149 GTTCCGCACTATATGGGCGAGATGCAAGGCGGCTGGCTCGGGCTC 198
 CC
 CC
 CC
 130 ----- 129
 CC
 CC
 CC
 199 GAGTCTCTGCTCCGGGCGGCAAAAGCGGGGAGGCTCAGAAAG 248
 CC
 CC
 CC
 130 ----- 129
 CC
 CC
 CC

19101

Listing for Sarah Pollock Sun Sep 21/17:13:34:1997

249 GAGCAGTCCCTCTGCTGAGCGACTGGGCTTTTCATGAAACAGAGAAG 298
 CC
 CC
 CC
 130 -----GCTTTTATGAAACAGAGAAG 150
 CC
 CC
 CC
 299 ATGGGCTGACGATTTTATTCAGAGCTTGCACAACTCTTATGCAATG 348
 CC
 CC
 CC
 151 ATGGGCTGACGATTTTATTCAGAGCTTGCACAACTCTTATGCAATG 200
 CC
 CC
 CC
 349 CAAACGTAAGTTTGGGTATTCGGGAAATTTGAAAGATCGTTTAAAT 398
 CC
 CC
 CC
 201 CAAAC----- 205
 CC
 CC
 CC
 399 TACTAGCAGCTAAACCTTGTTCACGTGGCAGGGGGTGGGGAGCTGAA 448
 CC
 CC
 CC
 206 ----- 205
 CC
 CC
 CC
 449 GTCTTTCTCCTCATGTAGATACCTGAGTTCATTCCTATTTGAAATCT 498
 CC
 CC
 CC
 206 -----ACGCTGAGTTCAGTCCATTTTGAATATGT 235
 CC
 CC
 CC
 499 CCCAACCTCAGAGCCGGAATTTATGAGCGCAACCTCTACCTCTCCCA 548
 CC
 CC
 CC
 236 CCCATCTCTCAGAGCCGAGCTTATGAGCGTAAACCTCTCTCTCCGCCA 285
 CC
 CC
 CC
 549 AGTCCCTCTCAACAAATCAACCTGGGTCTCAATTCCTCAATTC 589
 CC
 CC
 CC
 286 AGTCCCTCTCAACAAATCAACCTGGGTCTCAATTCCTCAATTC 326
 CC
 CC
 CC
 to: AA451285 from 1 to 420
 CCM101_19101 x AA451285
 121 COTCGCGCTGCGCCGCTGTGCGCCCTCGGTCCGATCTATGGCGGAT 170
 CC
 CC
 CC
 1 CGTCCGCTGCGCCGCTGTGCGCCCTCGGTCCGATCTATGGCGGAT 50
 CC
 CC
 CC
 171 GAGGGCGGCTGGCTGGGCTGGGCTGGGCTGGGCTGGGCTGGGCTGGG 220
 CC
 CC
 CC
 51 GCAGGGCGGCTGGCTGGGCTGGGCTGGGCTGGGCTGGGCTGGGCTGG 100
 CC
 CC
 CC
 221 ACAAAGCGGGGAGGCTCAGAAAGGAGGAGTCCGCTCTCTGAGC 270
 CC
 CC
 CC
 101 ACAAAGCGGGGAGGCTCAGAAAGGAGGAGTCCGCTCTCTGAGC 150
 CC
 CC
 CC
 271 GAGCTGGCTTTTCATGAAACAGAGAGGATGGGCTGAAACGATTTTATTC 320
 CC
 CC
 CC
 151 GAGT-GGCTTTTATGAAACAGAGAGGATGGGCTGAAACGATTTTATTC 199
 CC
 CC
 CC
 321 AGAAGCTTGCACAACTCTATGATGCAACGTAAGTTTGGGTATTC 370
 CC
 CC
 CC
 200 AGAAGTTGCCAGCAACCTATGATGCAAC----- 232
 CC
 CC
 CC
 371 GGGGAAATTTGAAAGATCGTTTAAATTTTACTAGCAGCTAAACCTTGTTC 420
 CC
 CC
 CC
 233 ----- 232
 CC
 CC
 CC
 421 ACGTGGCGAGGGGGTGGGGAGGCTGAAAGTCTTCTCTCCATGATAGTAC 470
 CC
 CC
 CC
 233 -----AC 234
 CC
 CC
 CC

19101

Listing for Sarah Pollock

Sun Sep 21 17:19:34 1997

[illegible]

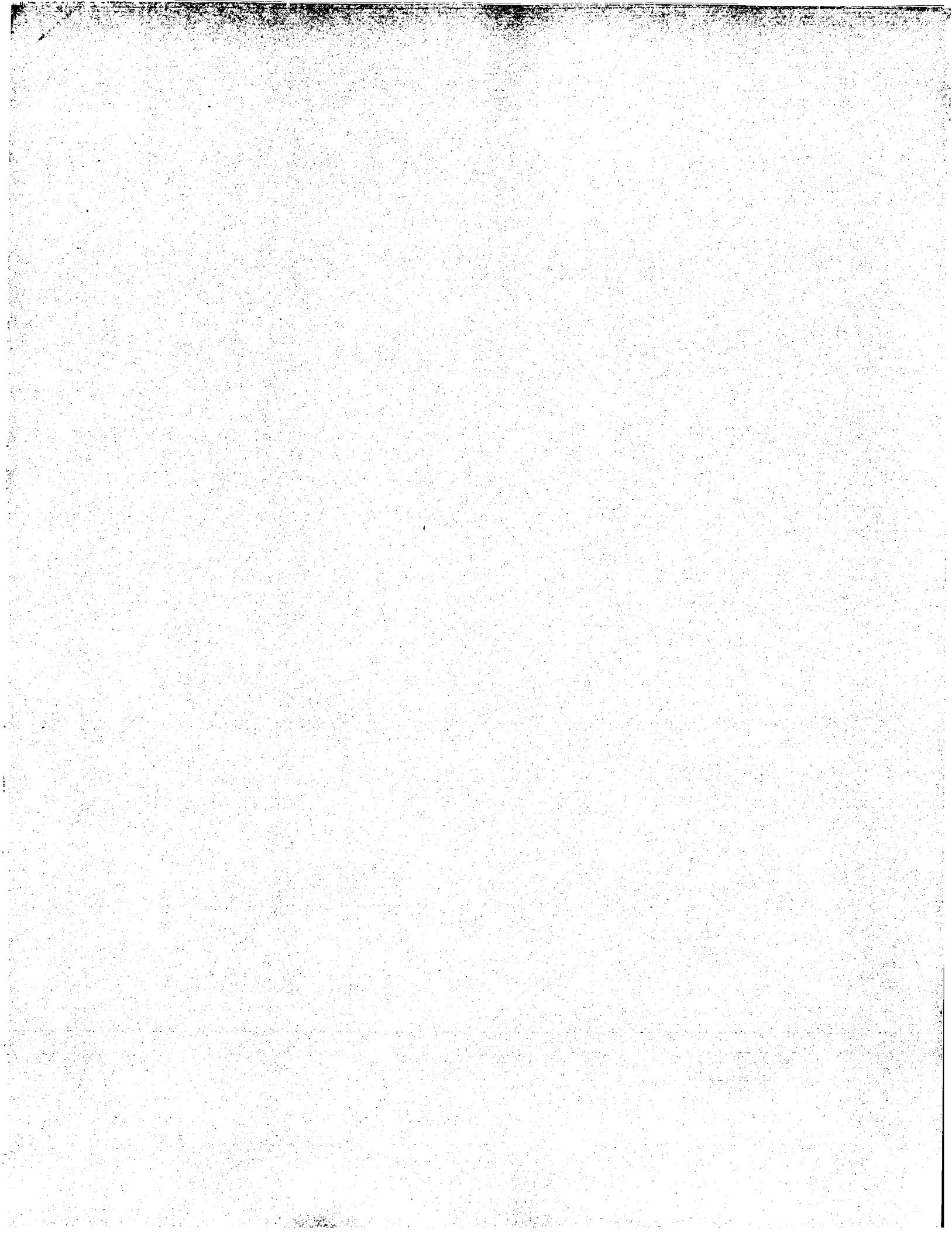
CCM101 19101 x W48549

Sequence 701 BP; 176 A; 194 C; 173 G; 158 T; 0 other:

11

19101

19101



B-26

Sequencing for Sarah Pollock

Seq Sep 21 14:13:57.1897

>CCM101_19101_0 #TI Brain
TCACTGCTCCTTTAGTCTCTTTGGGACACTTTCCAAATATCGGGCGATGACCGTCAAAAC
CGAGGCTGCTCCGAAGTACCTCACCCTACTCCAGATGAGGGGAATGGTAGCAATCCTCAT
CGCTTTCAGAAACAGAGAGAGGTGGCCCTGACGATTTTATTCAGAGCTTSCCAACAA
CTCCTATGCATGCAACACCCCTGAAGTCAATCCTATTGAAATCTCCCAACCTCAGGA
GCCCGAAGTTATGAAAGCCCAACCCCTCACCCTCCCAAGTCCCTCTCAACAAATCAACCT
GGGTCCATCTCAAAATCCCAACGCCAAACCCCTCGACTTCCACTTCTTGAAGTATCGG
AAAGGCAGTTTTGGAAAGTTCTTCTAGCAGGCACACAGGCAGAGAGCATTTCTATGC
CGTCAAGTT
>CCM101_19101_1
TCACTGCTCCTTTAGTCTCTTTGGGACACTTTCCAAATATCGGGCGATGACCGTCAAAAC
CGAGGCTGCTCCGAAGTACCTCACCCTACTCCAGATGAGGGGAATGGTAGCAATCCTCAT
CGCTTTCAGAAACAGAGAGAGGTGGCCCTGACGATTTTATTCAGAGCTTSCCAACAA
CTCCTATGCATGCAACACCCCTGAAGTCAATCCTATTGAAATCTCCCAACCTCAGGA
TTTACTAGCAGCTAAACCTTGTTCACGGTGGCGAGGGGTGGGGAGCTGAAGTCTTTCC
TCCATGCTAGAT

19101 transcript

